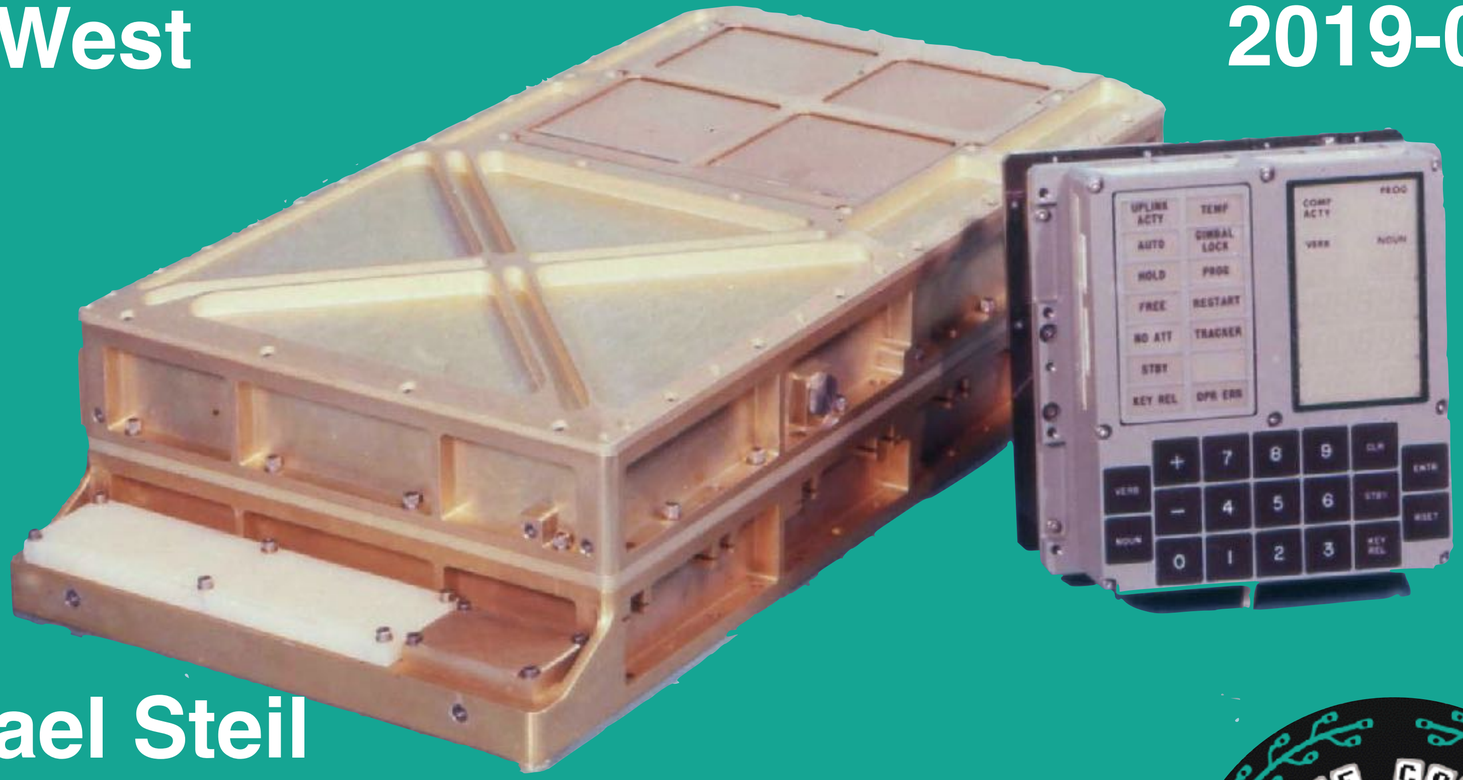
VCF West

2019-08-04

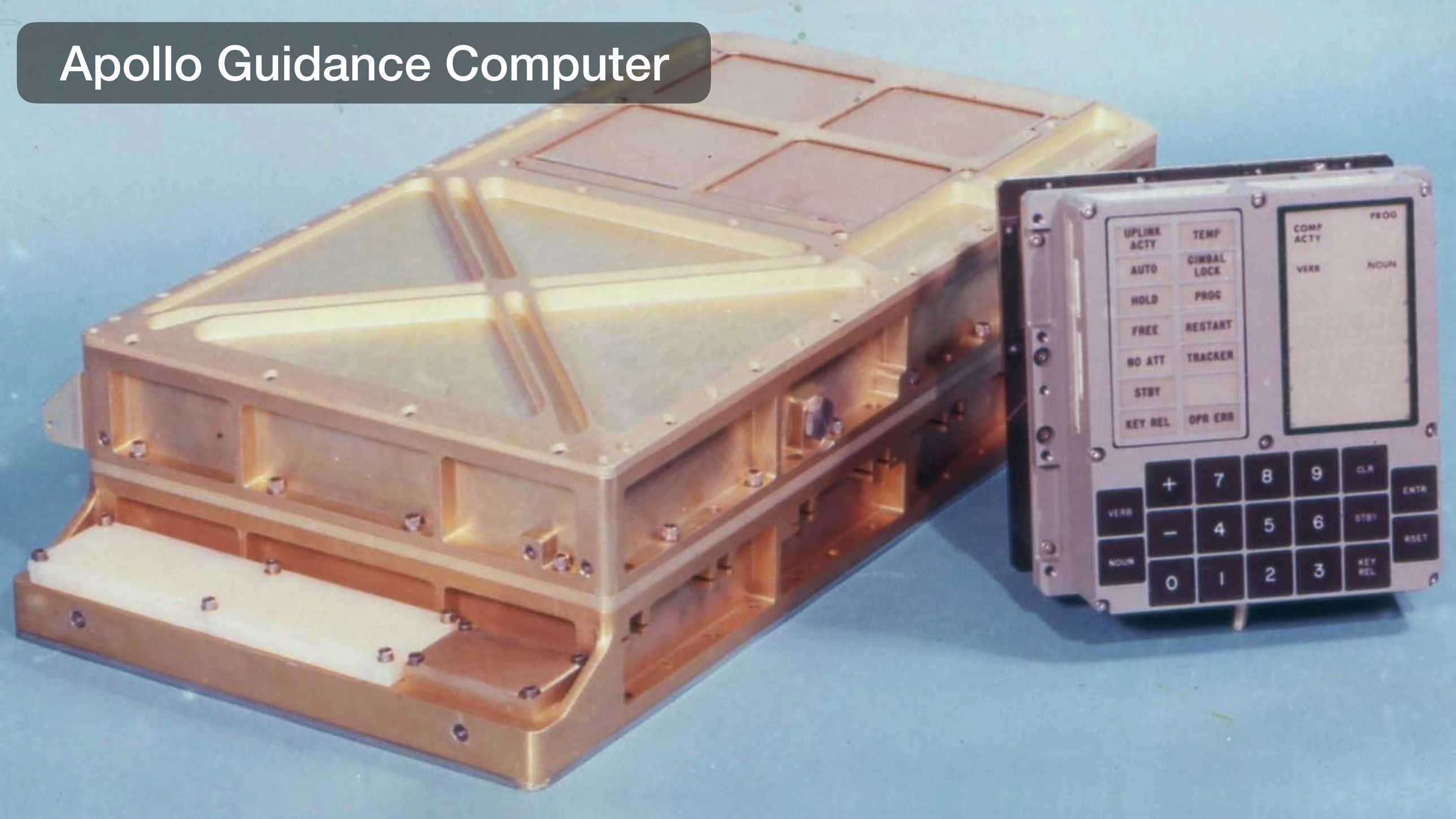


Michael Steil

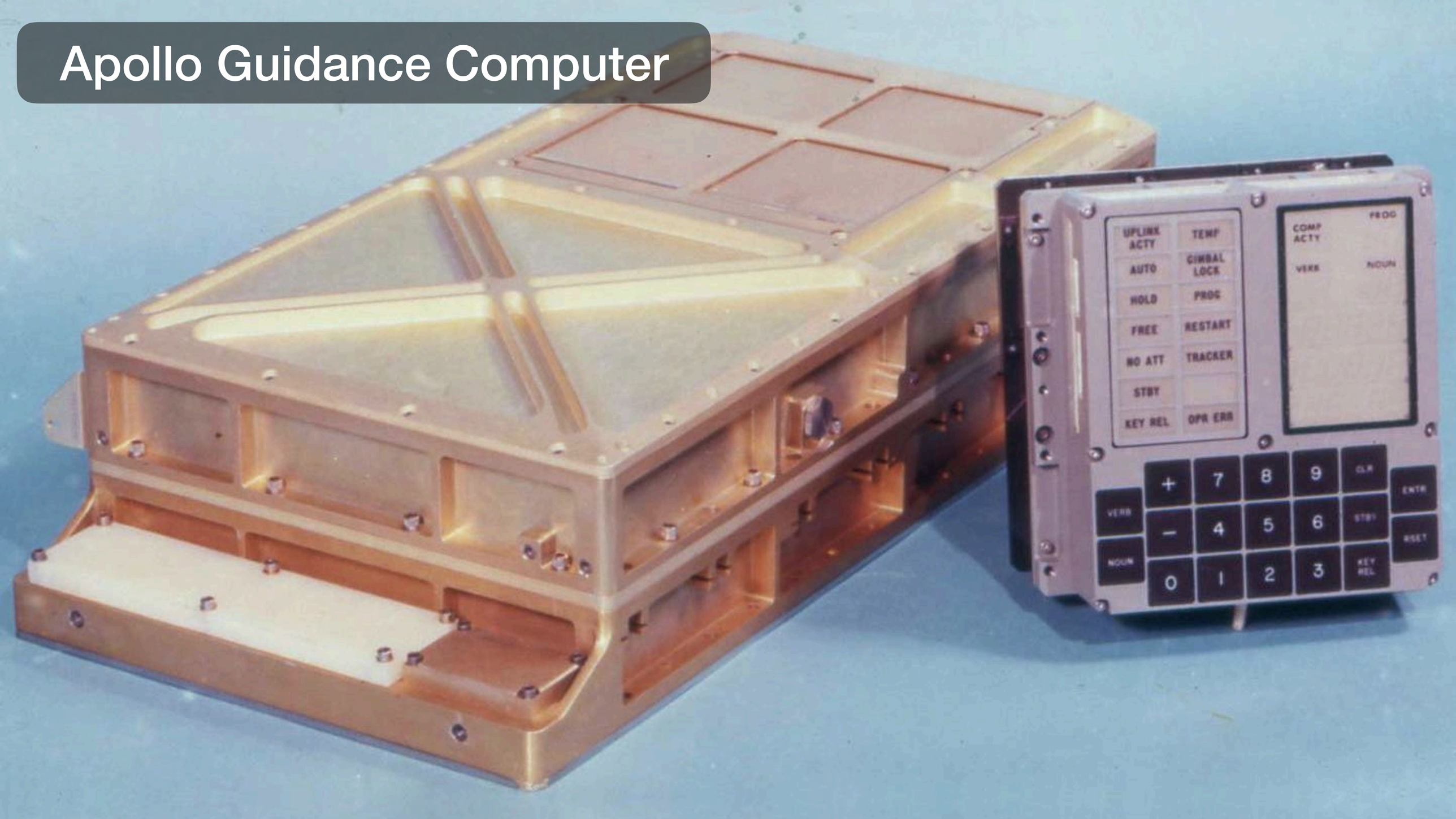
The Apollo Guidance Computer

@pagetable
http://www.pagetable.com/

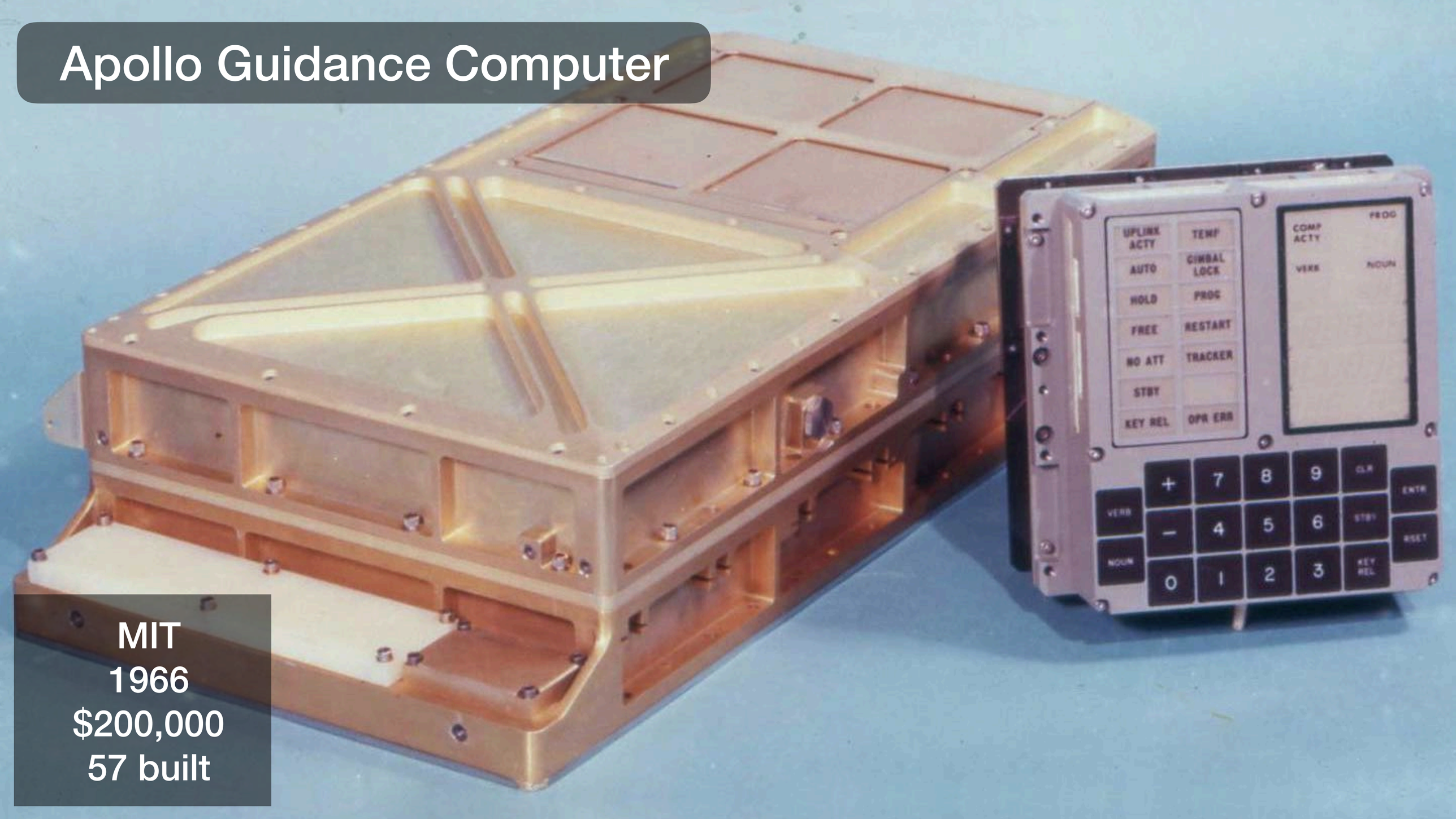
Apollo Guidance Computer



Apollo Guidance Computer

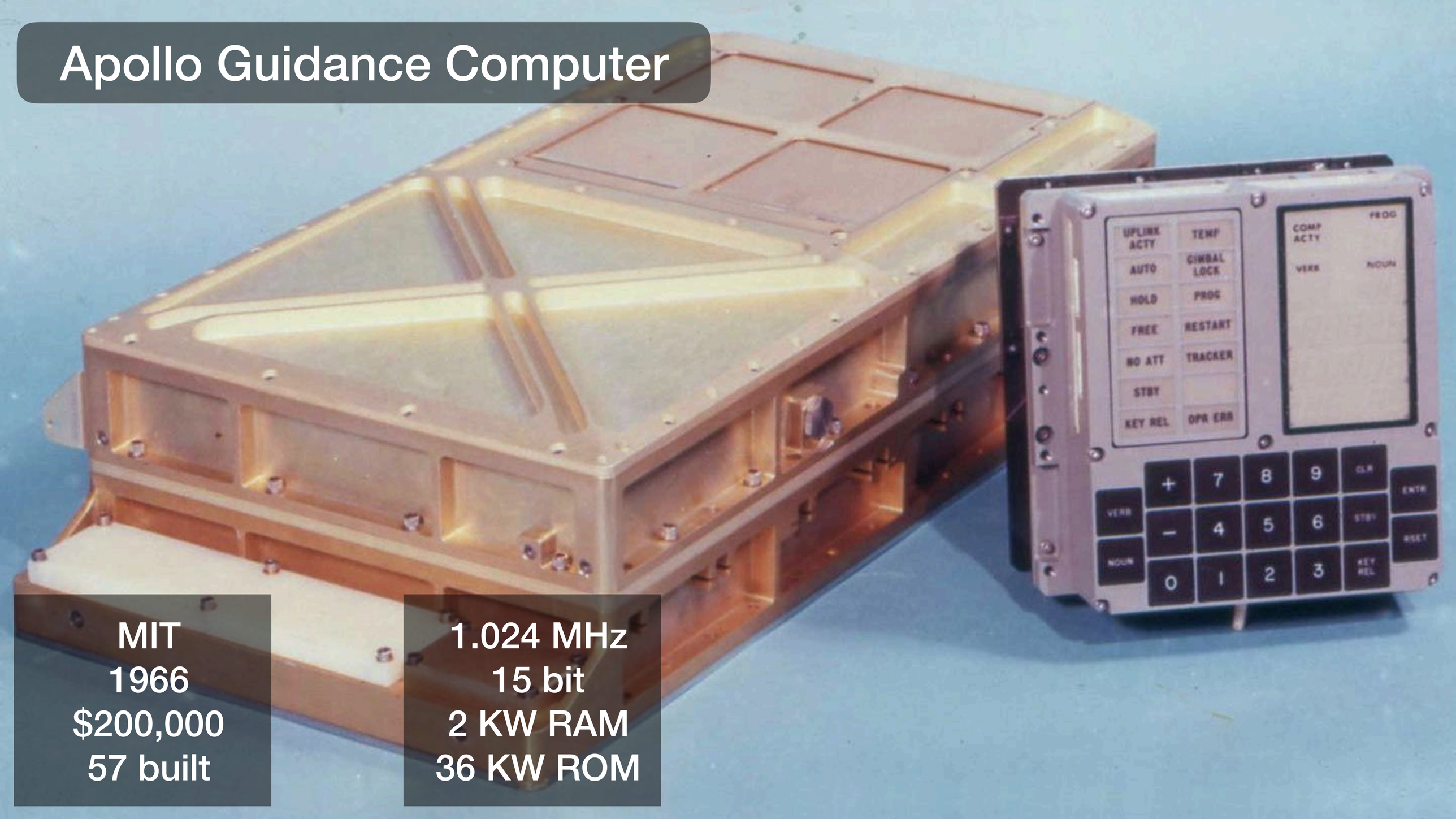


Apollo Guidance Computer



MIT
1966
$200,000
57 built

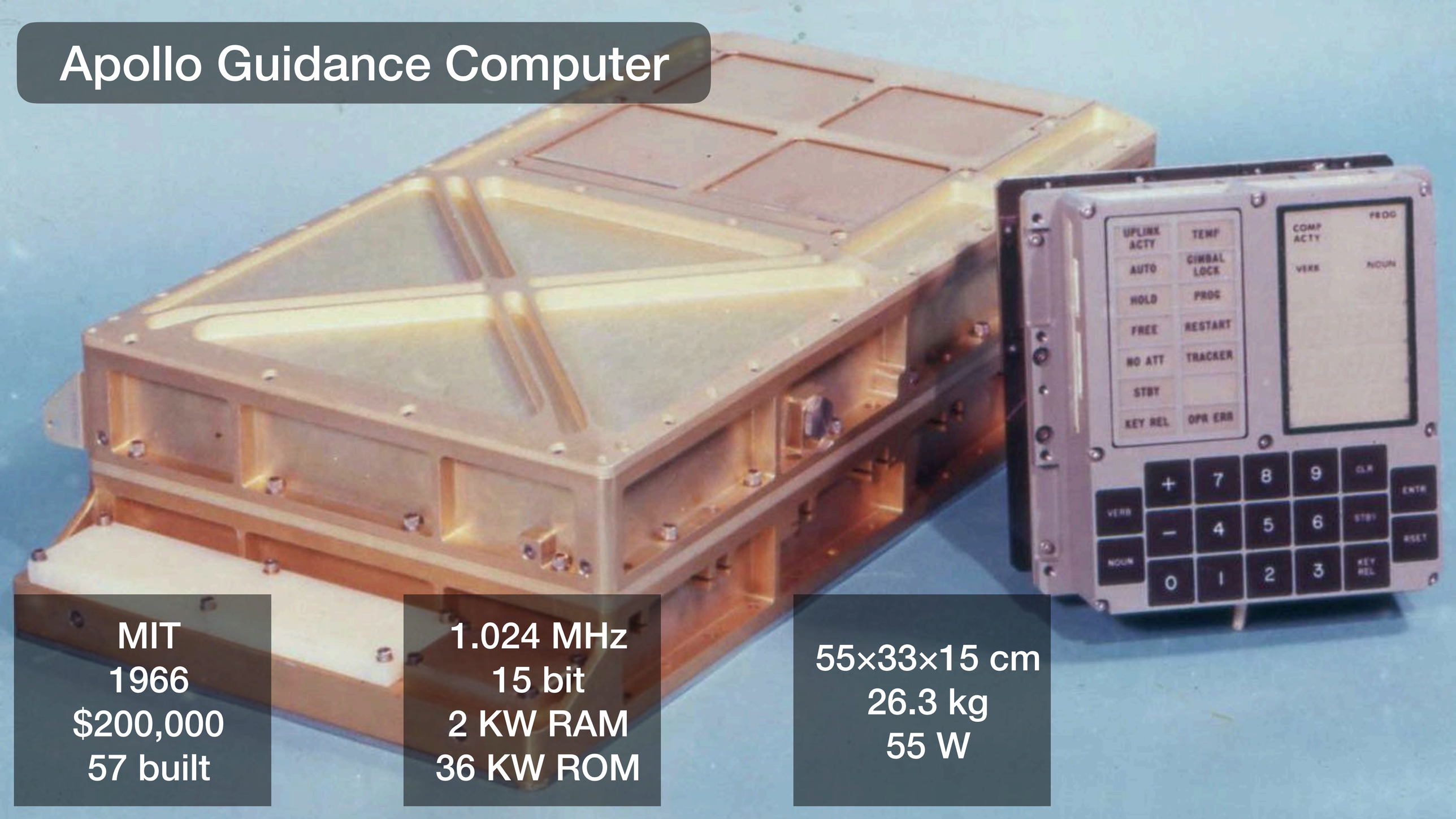
Apollo Guidance Computer



MIT
1966
$200,000
57 built

1.024 MHz
15 bit
2 KW RAM
36 KW ROM

Apollo Guidance Computer

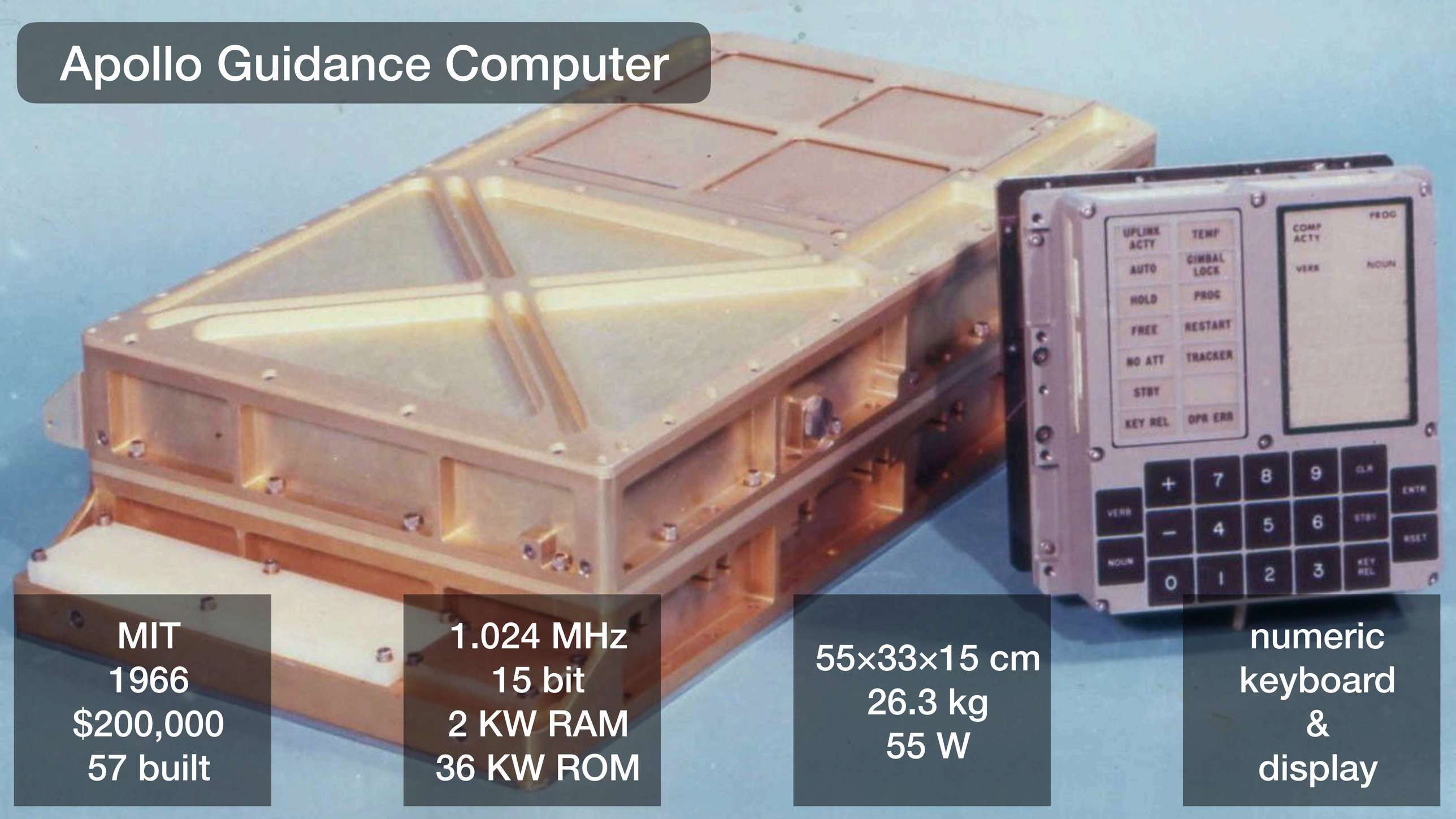


MIT
1966
$200,000
57 built

1.024 MHz
15 bit
2 KW RAM
36 KW ROM

55×33×15 cm
26.3 kg
55 W

Apollo Guidance Computer



MIT
1966
$200,000
57 built

1.024 MHz
15 bit
2 KW RAM
36 KW ROM

55×33×15 cm
26.3 kg
55 W

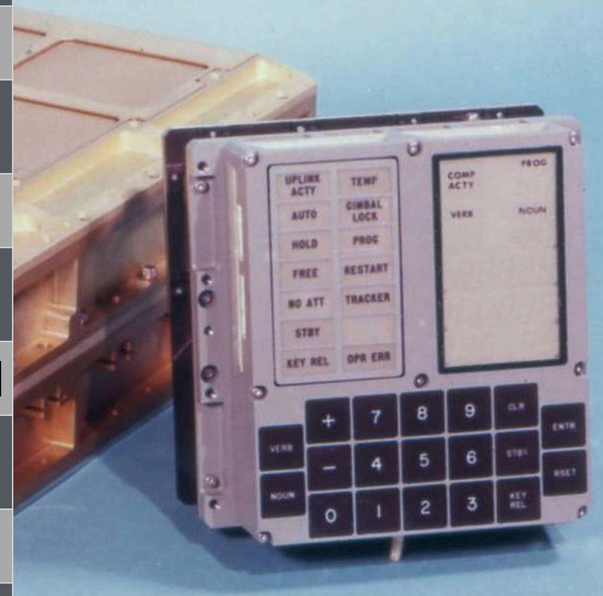
numeric
keyboard
&
display

DEC PDP-8

1965

Transistors	
10,000	16,800
Memory Speed	
600 KHz	85 KHz
Word Size	
12 bit	15 bit
Kilo-Adds per Second	
384	44
Kilo-Muls per Second	
0.3	22
Weight	
82 kg	32 kg
Power Consumption	
400 W	55 W

Apollo Guidance Computer

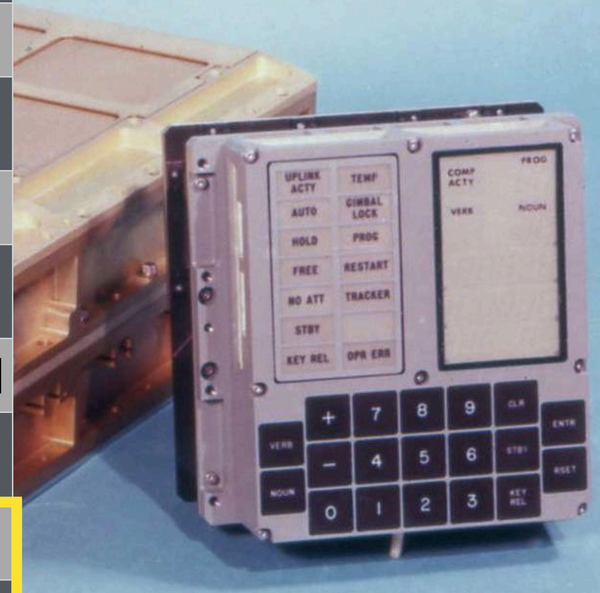


1966

DEC PDP-8

1965

Apollo Guidance Computer



Transistors	
10,000	16,800
Memory Speed	
600 KHz	85 KHz
Word Size	
12 bit	15 bit
Kilo-Adds per Second	
384	44
Kilo-Muls per Second	
0.3	22
Weight	
82 kg	32 kg
Power Consumption	
400 W	55 W

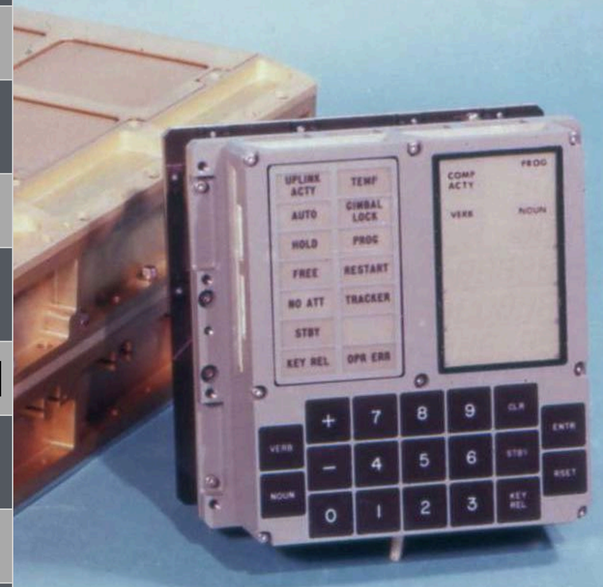
1966

UNIVAC I

1951

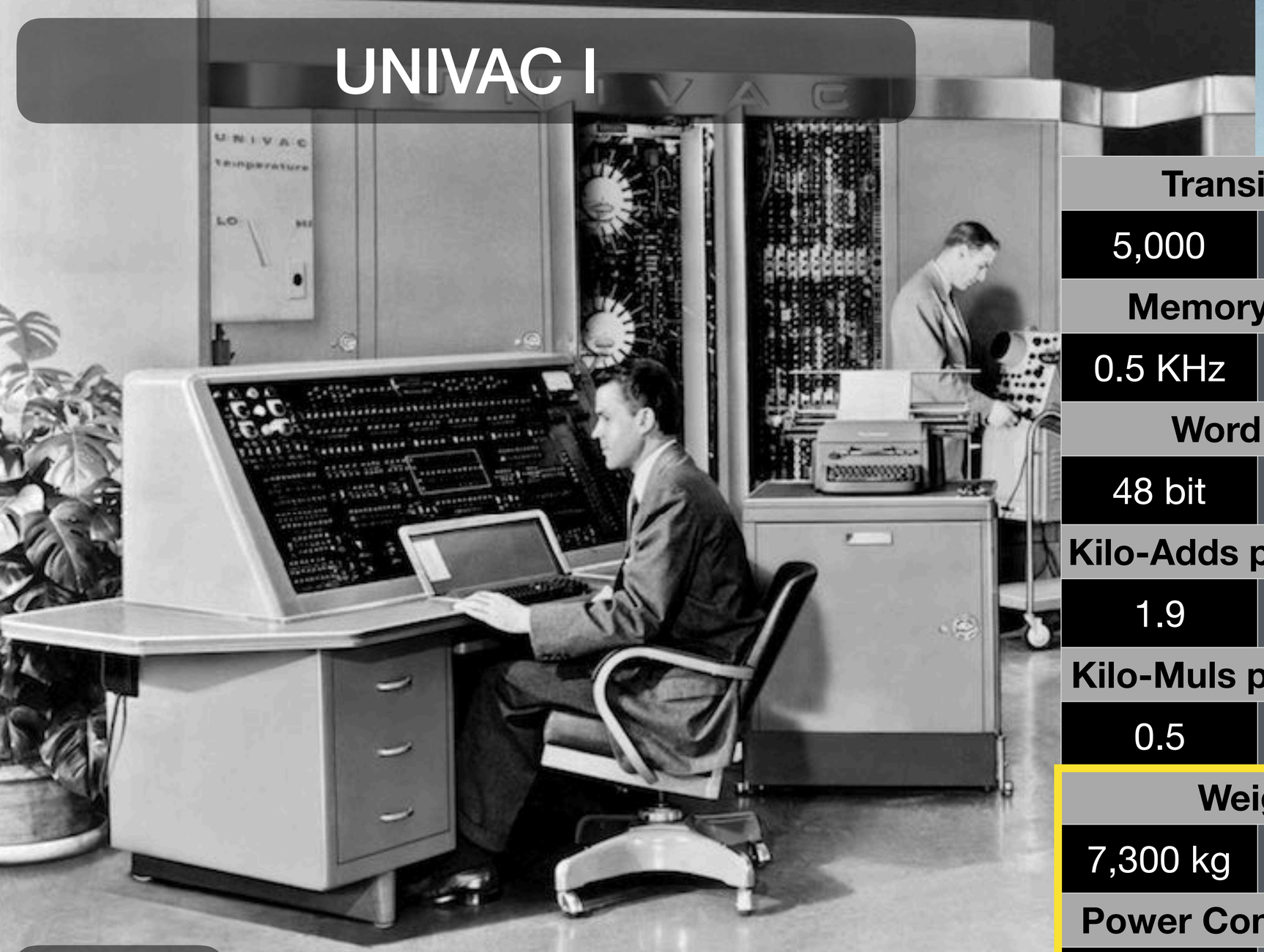
Transistors	
5,000	16,800
Memory Speed	
0.5 KHz	85 KHz
Word Size	
48 bit	15 bit
Kilo-Adds per Second	
1.9	44
Kilo-Muls per Second	
0.5	22
Weight	
7,300 kg	32 kg
Power Consumption	
125,000 W	55 W

Apollo Guidance Computer



1966

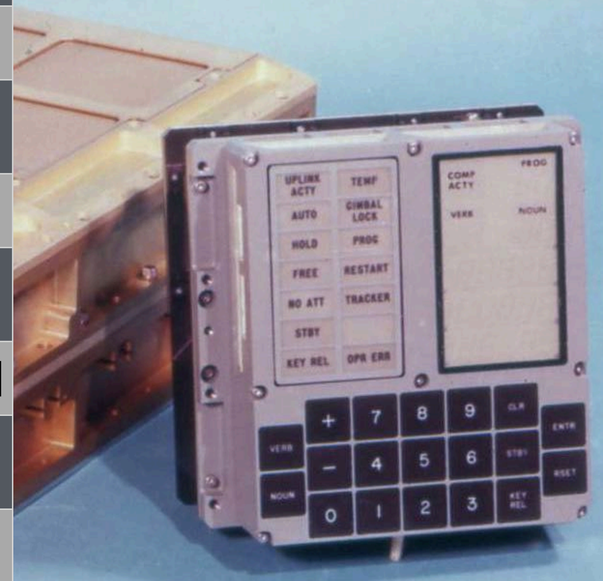
UNIVAC I



1951

Transistors	
5,000	16,800
Memory Speed	
0.5 KHz	85 KHz
Word Size	
48 bit	15 bit
Kilo-Adds per Second	
1.9	44
Kilo-Muls per Second	
0.5	22
Weight	
7,300 kg	32 kg
Power Consumption	
125,000 W	55 W

Apollo Guidance Computer



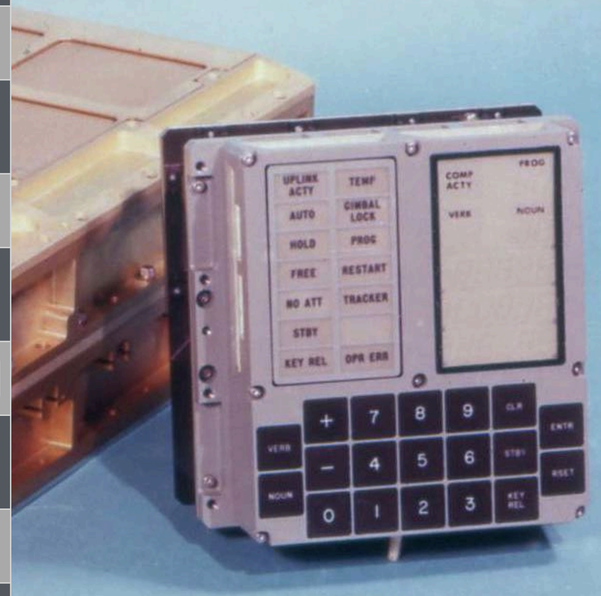
1966

Commodore PET

1977

Transistors	
3,510	16,800
Memory Speed	
1000 KHz	85 KHz
Word Size	
8 bit	15 bit
Kilo-Adds per Second	
500	44
Kilo-Muls per Second	
3	22
Weight	
1 kg	32 kg
Power Consumption	
10 W	55 W

Apollo Guidance Computer

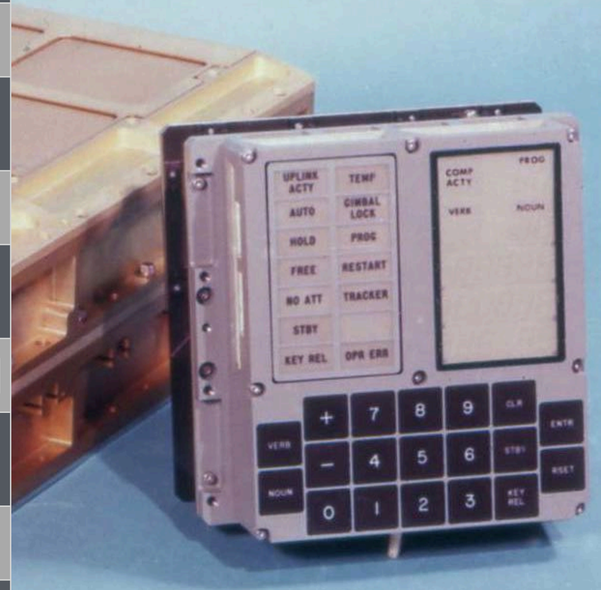


1966

Commodore PET

Apollo Guidance Computer

Transistors	
3,510	16,800
Memory Speed	
1000 KHz	85 KHz
Word Size	
8 bit	15 bit
Kilo-Adds per Second	
500	44
Kilo-Muls per Second	
3	22
Weight	
1 kg	32 kg
Power Consumption	
10 W	55 W



1977

1966

15 bit

One’s Complement

Accumulator Machine

very 60s
very alien

15 bit

One’s Complement

Accumulator Machine

very 60s
very alien

Integrated Circuits

Microcoded

Core Rope Memory

innovative
for its time

15 bit	One’s Complement	Accumulator Machine	very 60s very alien
Integrated Circuits	Microcoded	Core Rope Memory	innovative for its time
Gyroscope Accelerometer	Radar	Jets	very interesting peripherals

15 bit	One’s Complement	Accumulator Machine	very 60s very alien
Integrated Circuits	Microcoded	Core Rope Memory	innovative for its time
Gyroscope Accelerometer	Radar	Jets	very interesting peripherals
Preemptive Real-Time	Fault-Tolerant	Virtual Machines	revolutionary for its time

15 bit	One’s Complement	Accumulator Machine	very 60s very alien
Integrated Circuits	Microcoded	Core Rope Memory	innovative for its time
Gyroscope Accelerometer	Radar	Jets	very interesting peripherals
Preemptive Real-Time	Fault-Tolerant	Virtual Machines	revolutionary for its time
Guidance	Navigation	Fly-By-Wire	can fly you to the moon


APOLLO
NASA

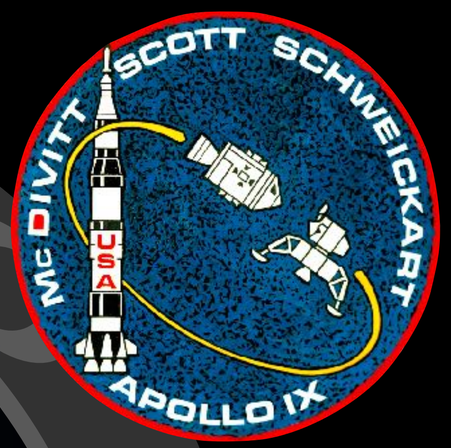

LM-1
APOLLO 5


APOLLO
NASA


LM-1
APOLLO 5

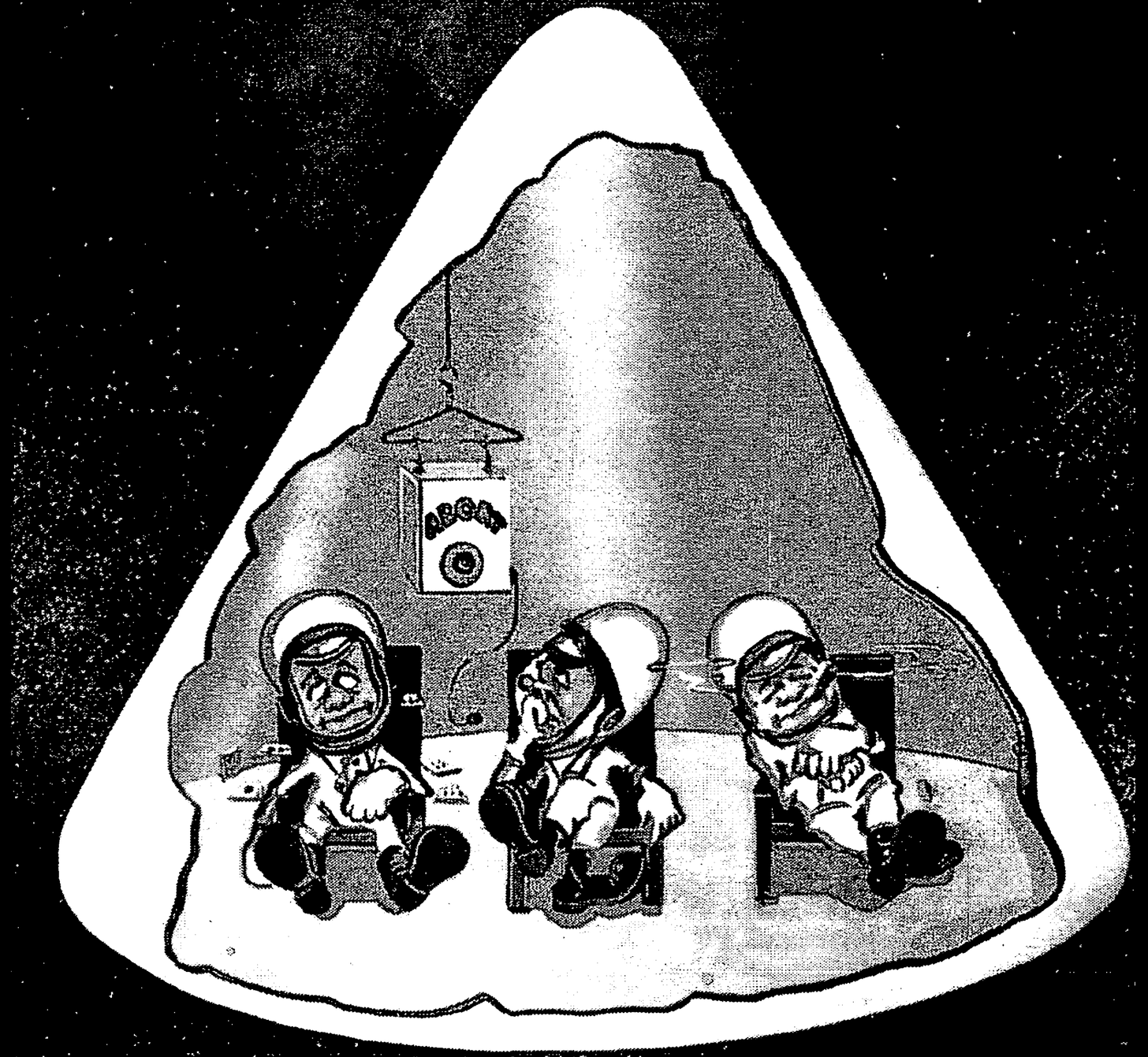

VII
SCHIRRA · EISELE · CUNNINGHAM

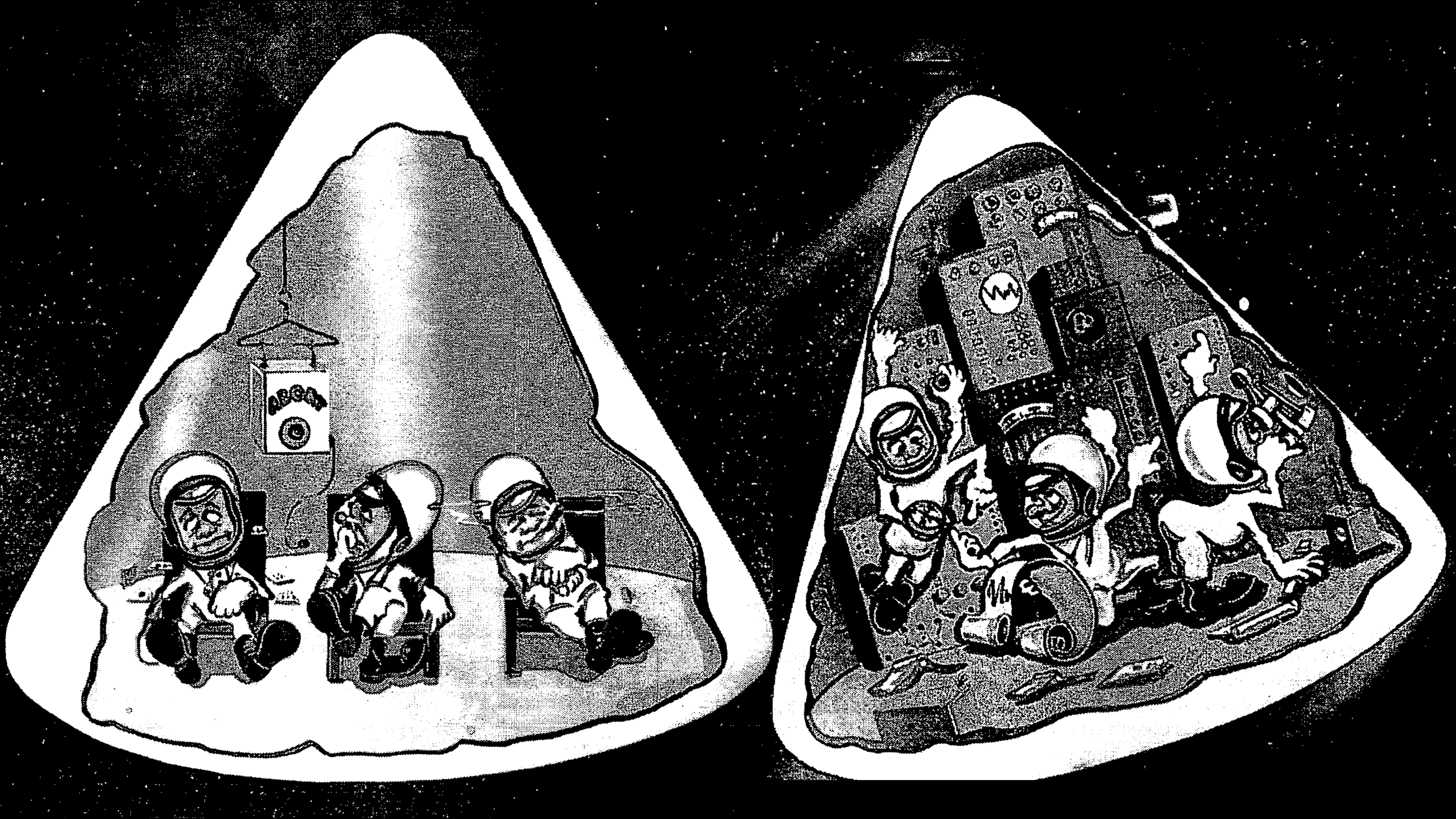

BORMAN LOVELL ANDERS

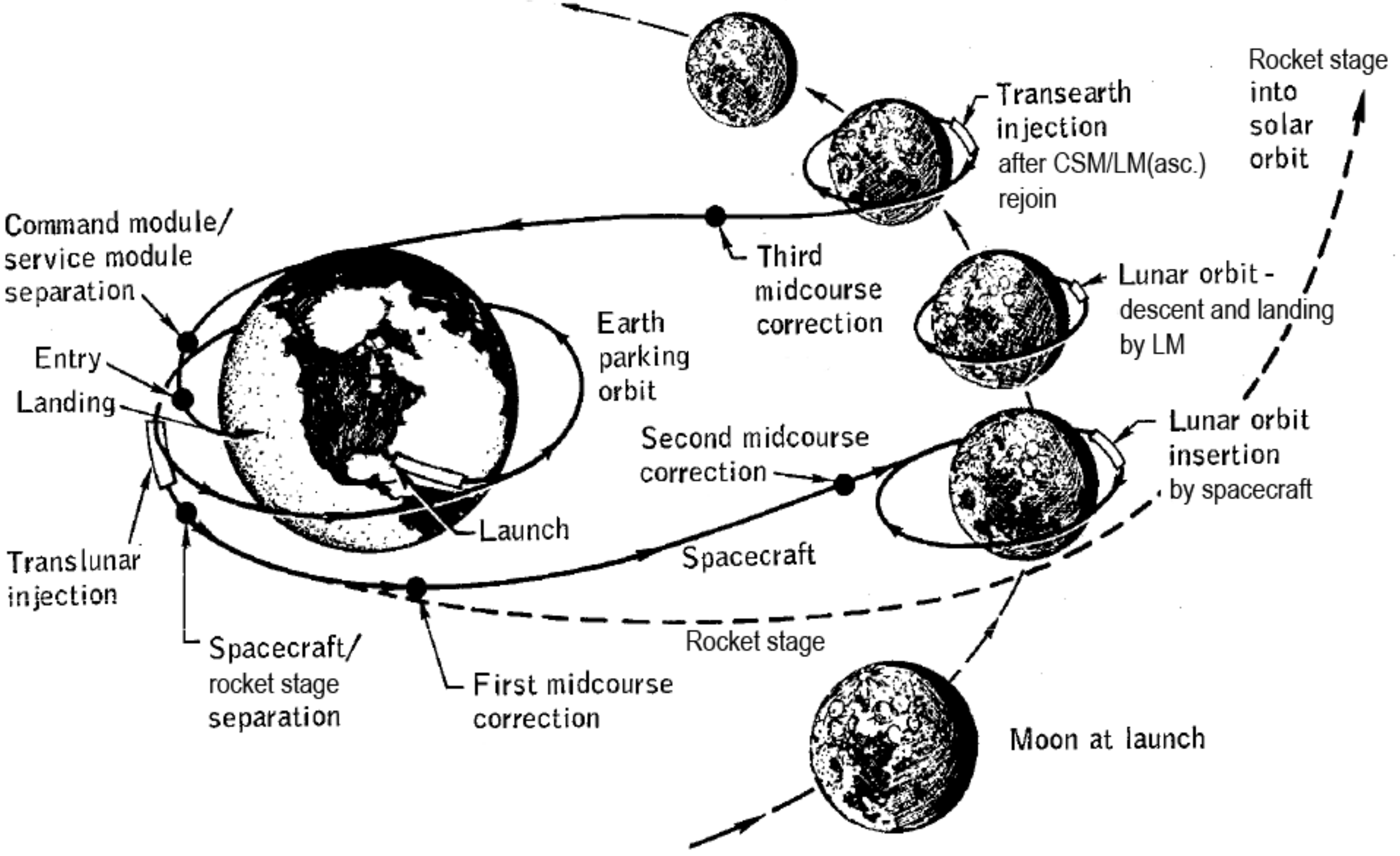

McDIVITT SCOTT SCHWEICKART
USA
APOLLO IX


APOLLO
STAFFORD YOUNG CERNAN

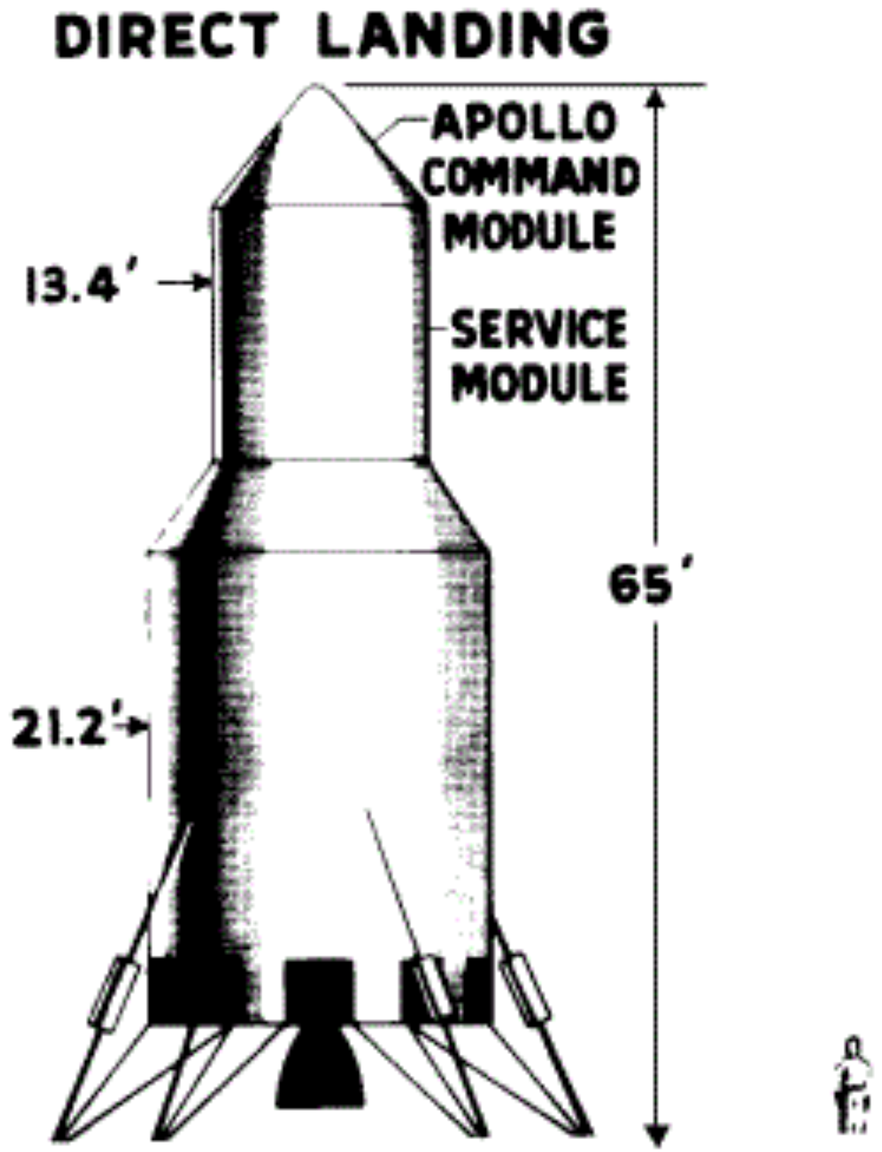

LM-1
APOLLO 5
VII
SCHIRRA · EISELE · CUNNINGHAM
BORMAN LOVELL ANDERS
McDIVITT SCOTT SCHWEICKART
USA
APOLLO IX
APOLLO
STAFFORD YOUNG CERNAN
APOLLO 11
APOLLO XII
Conrad · Gordon · Bean
APOLLO XIII
EX LUNA, SCIENTIA
APOLLO 14
Shepard
Roosa
Mitchell
APOLLO 15
SCOTT WORDEN IRWIN
APOLLO 16
YOUNG · MATTINGLY · DUKE
APOLLO XVII
CERNAN · EVANS · SCHMITT


ABCD

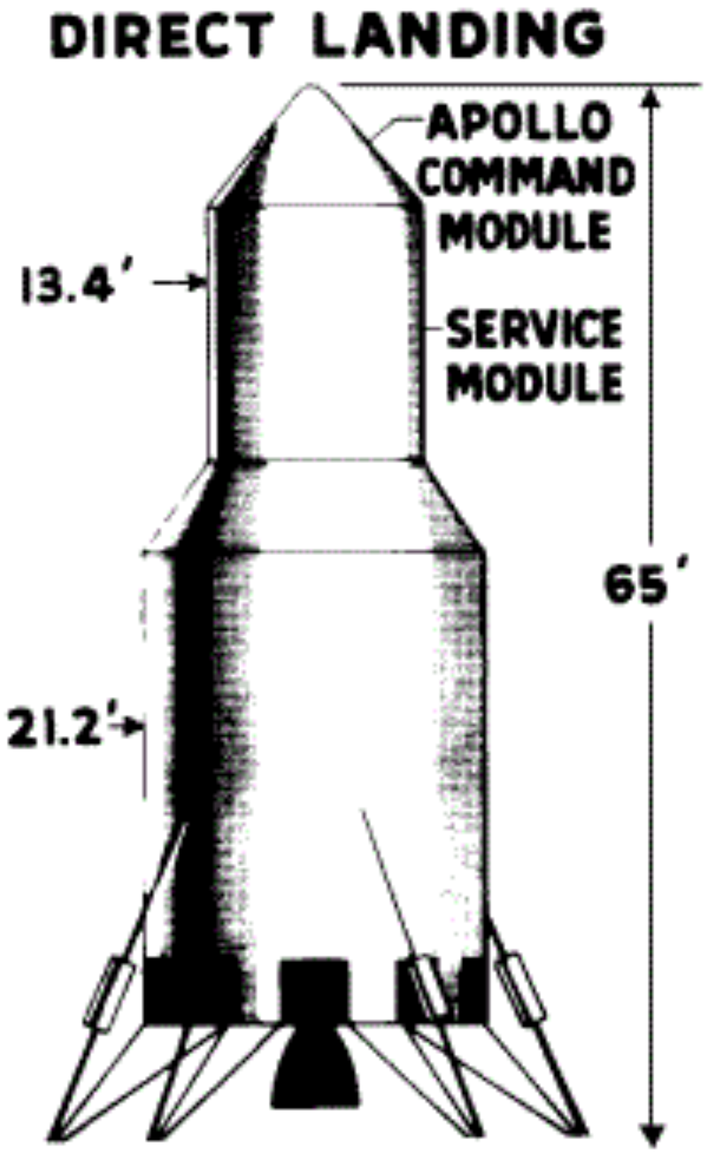

ABC


Rocket stage into solar orbit
Transearth injection after CSM/LM(asc.) rejoin
Command module/ service module separation
Third midcourse correction
Lunar orbit - descent and landing by LM
Entry
Landing
Earth parking orbit
Second midcourse correction
Lunar orbit insertion by spacecraft
Launch
Translunar injection
Spacecraft
Rocket stage
Spacecraft/ rocket stage separation
First midcourse correction
Moon at launch

Mission

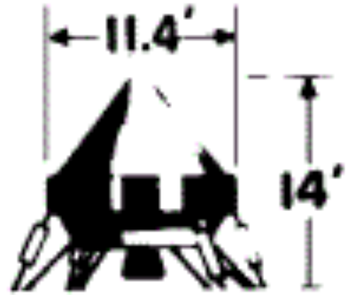

DIRECT LANDING
APOLLO COMMAND MODULE
SERVICE MODULE
13.4'
21.2'
65'

Mission



LUNAR FERRY OF LUNAR RENDEZVOUS

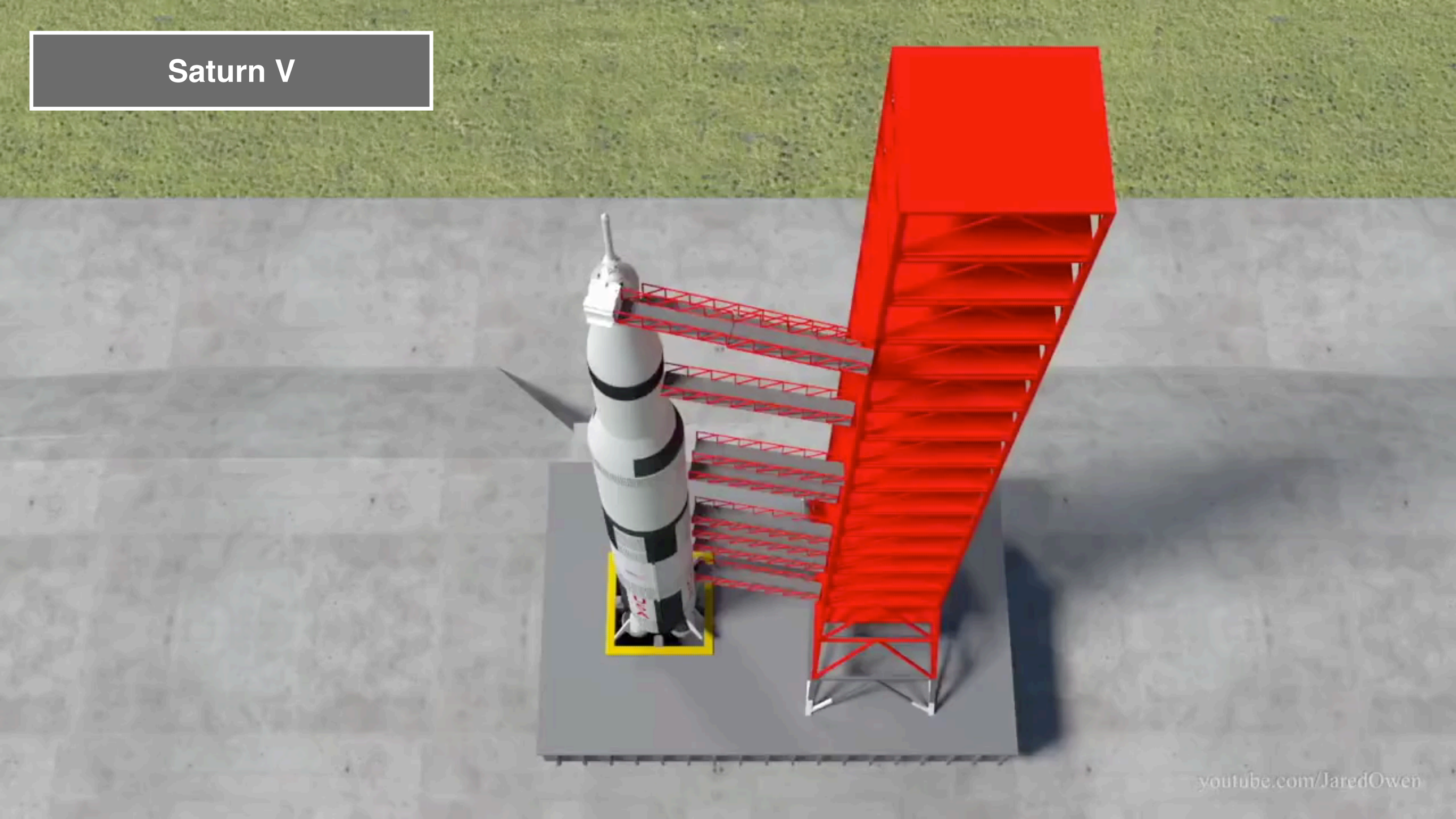
LUNAR EXCURSION VEHICLE



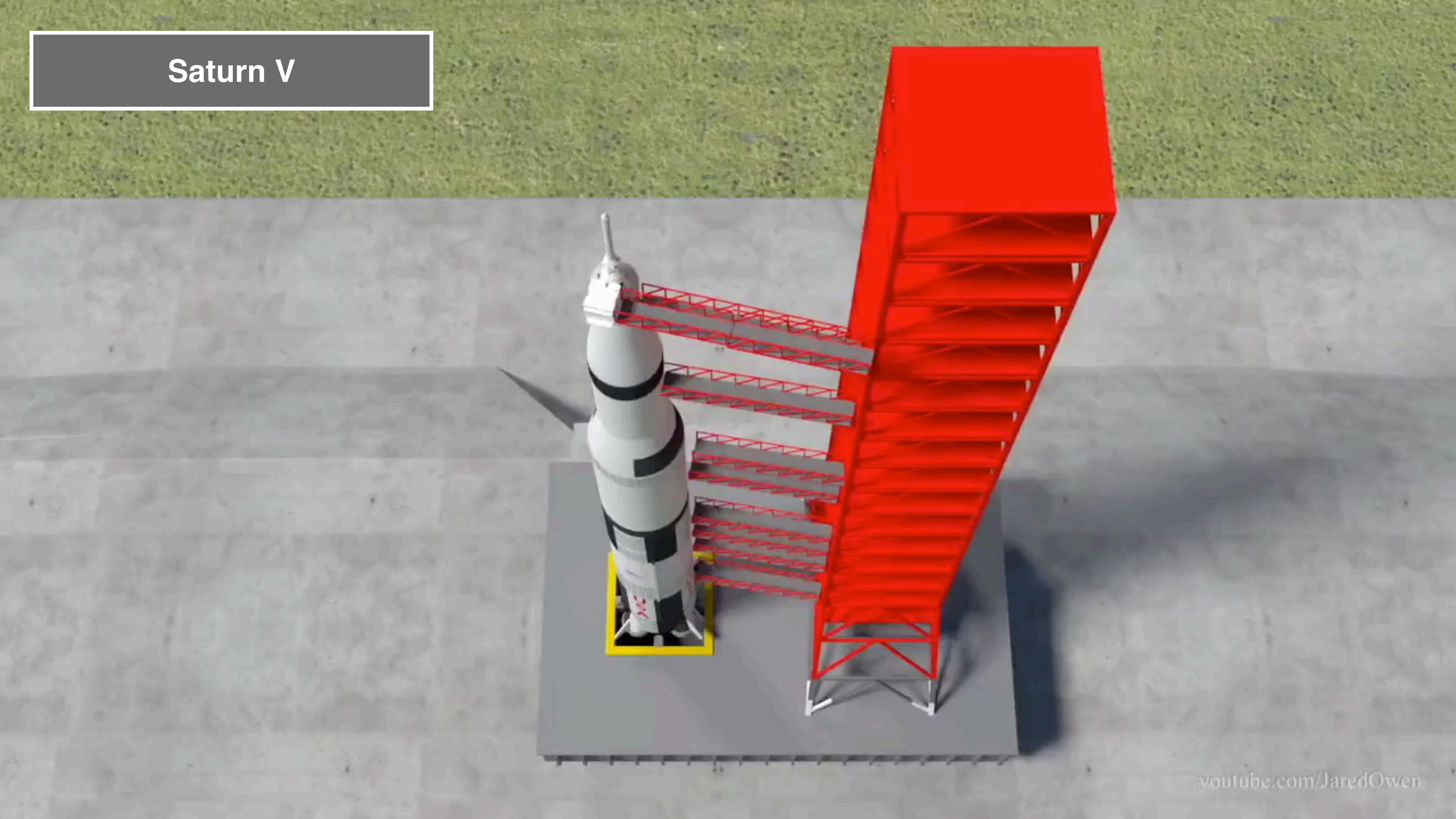
Apollo Spacecraft

Apollo Spacecraft

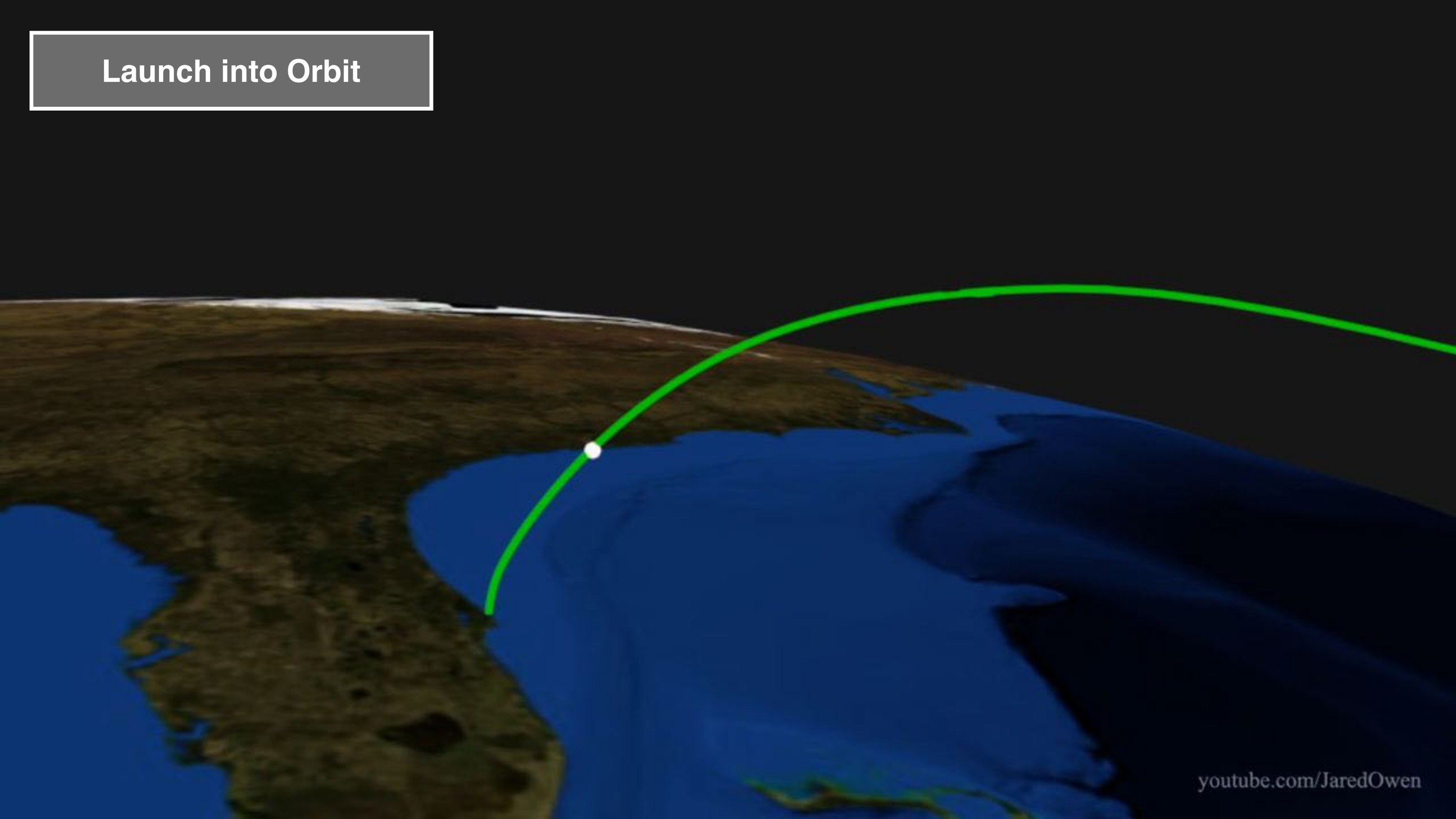
Saturn V



Saturn V



Launch into Orbit

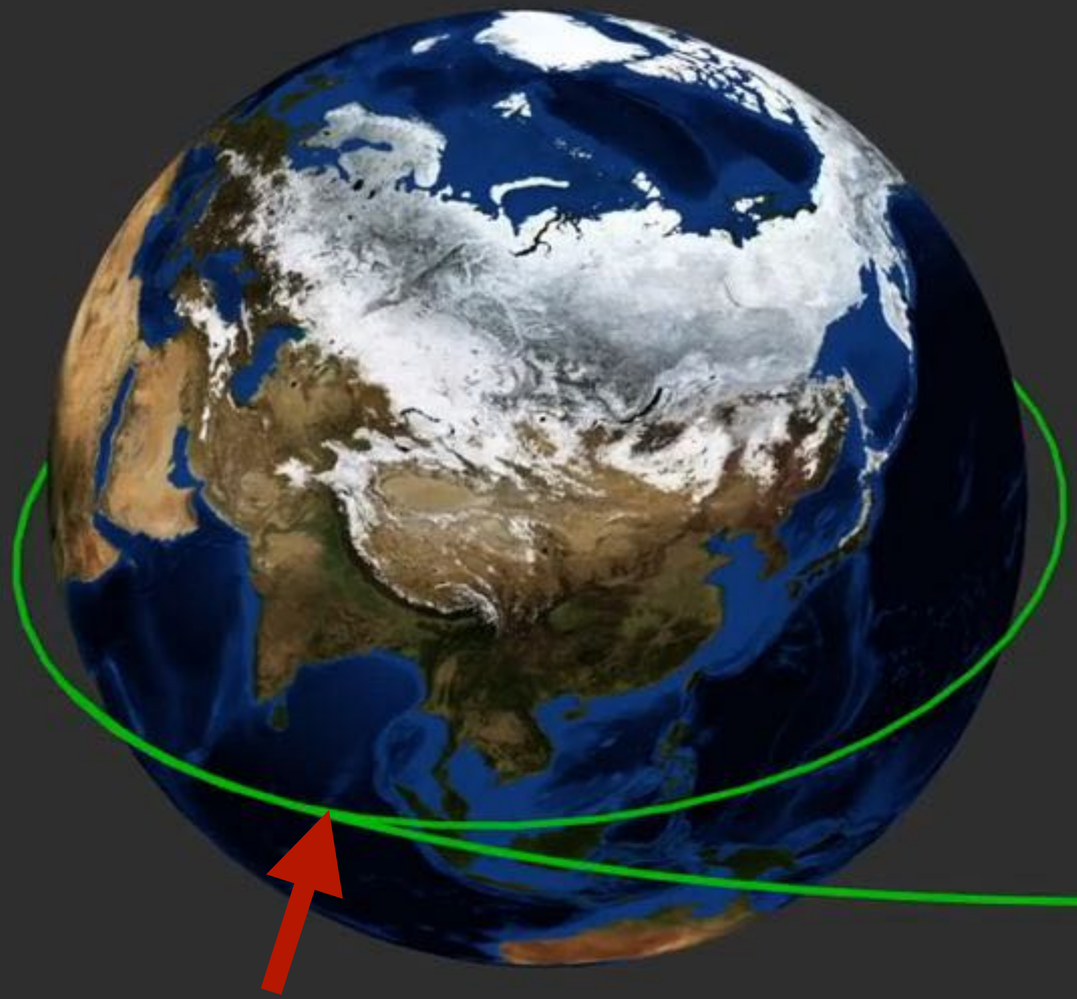


Earth Orbit

Trans-Lunar Injection

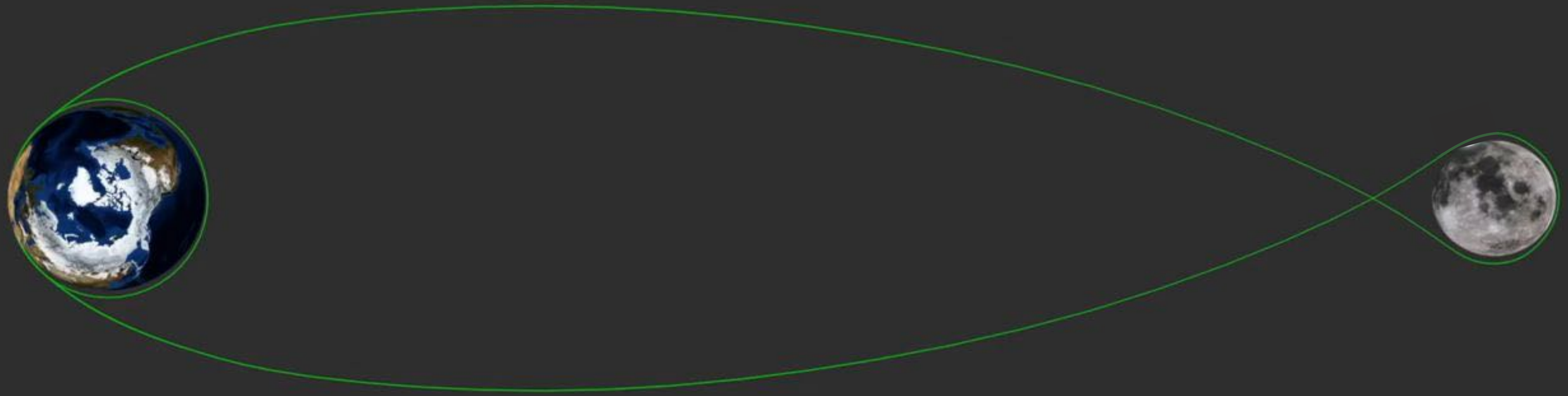
youtube.com/JaredOwen

Trans-Lunar Injection



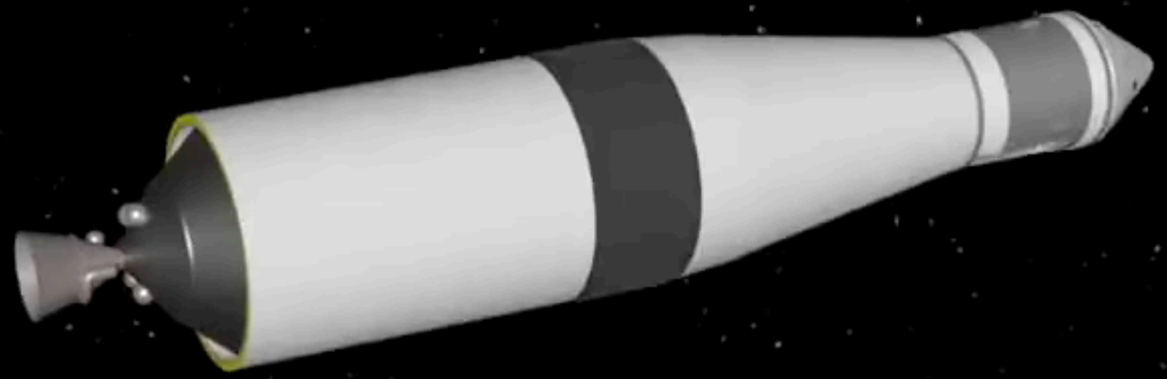
youtube.com/JaredOwen

Free Return Orbit



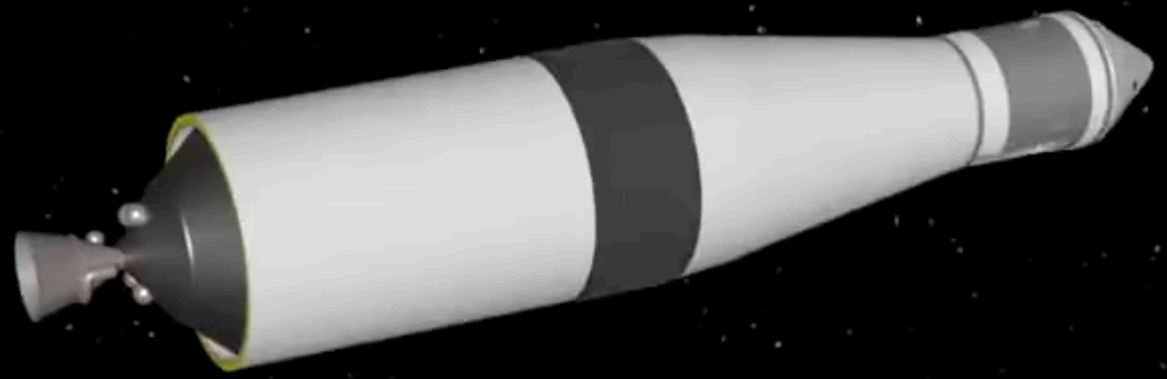
youtube.com/JaredOwen

Transposition & Docking



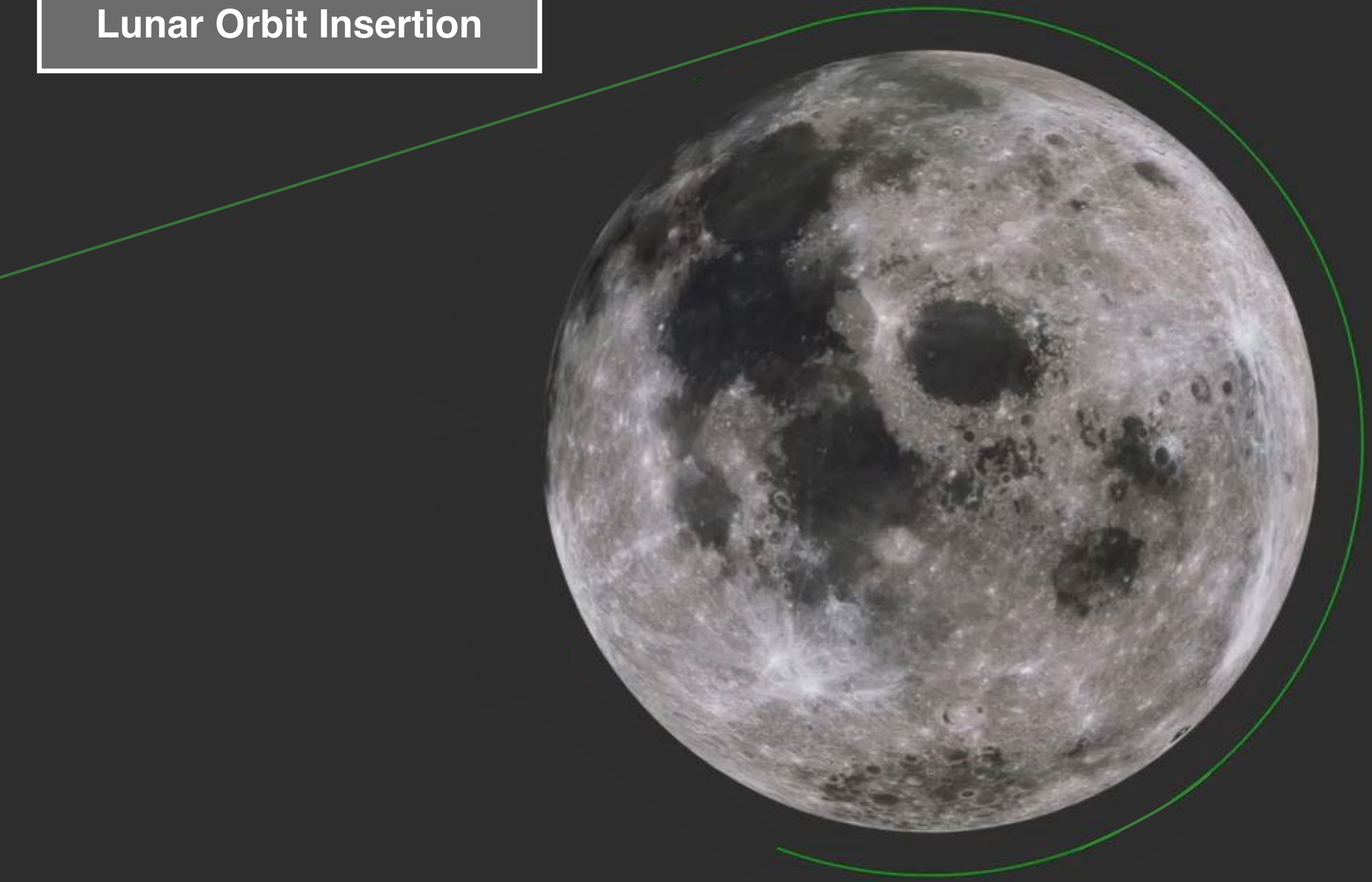
youtube.com/JaredOwen

Transposition & Docking



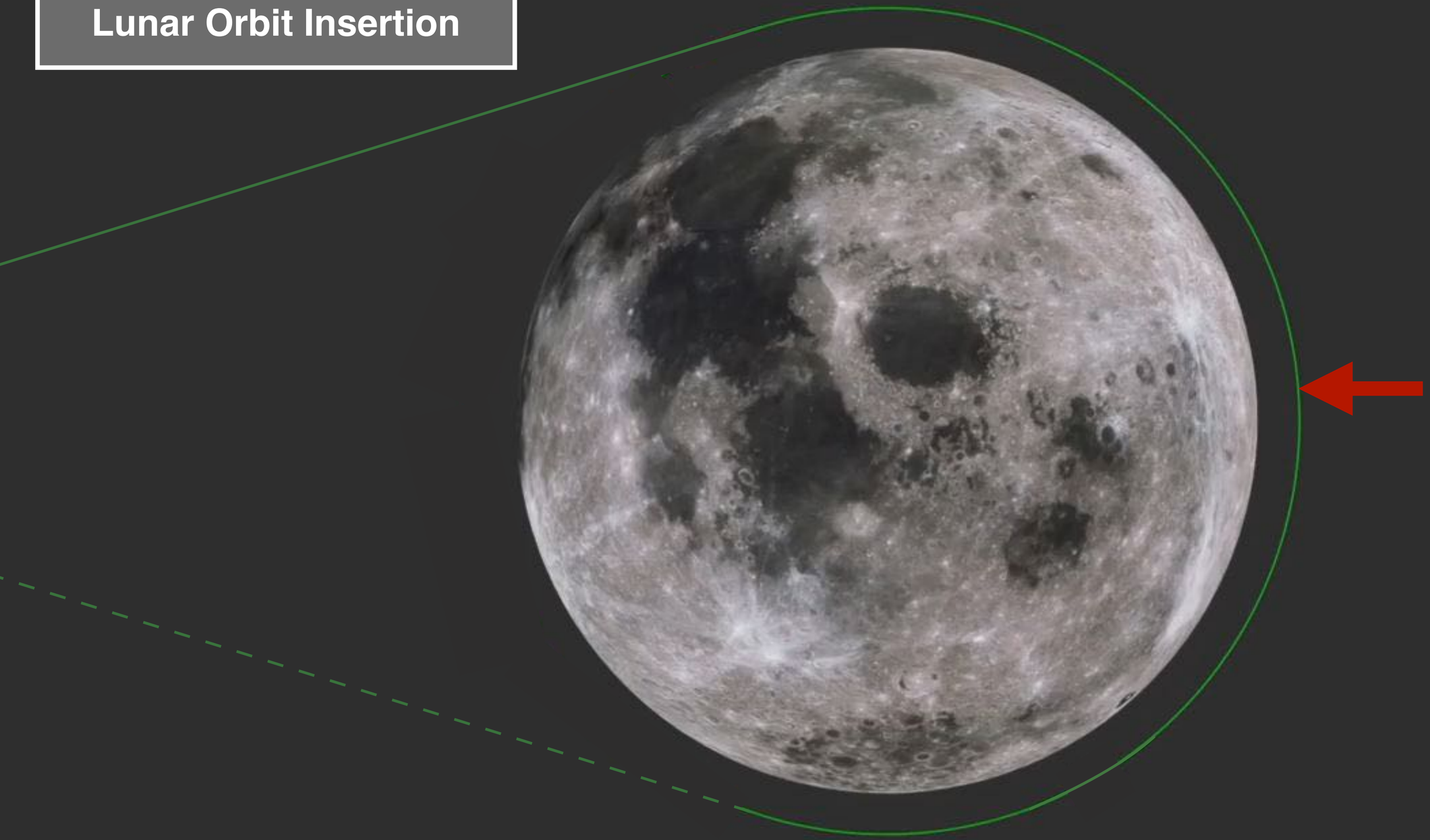
youtube.com/JaredOwen

Lunar Orbit Insertion



youtube.com/JaredOwen

Lunar Orbit Insertion



youtube.com/JaredOwen

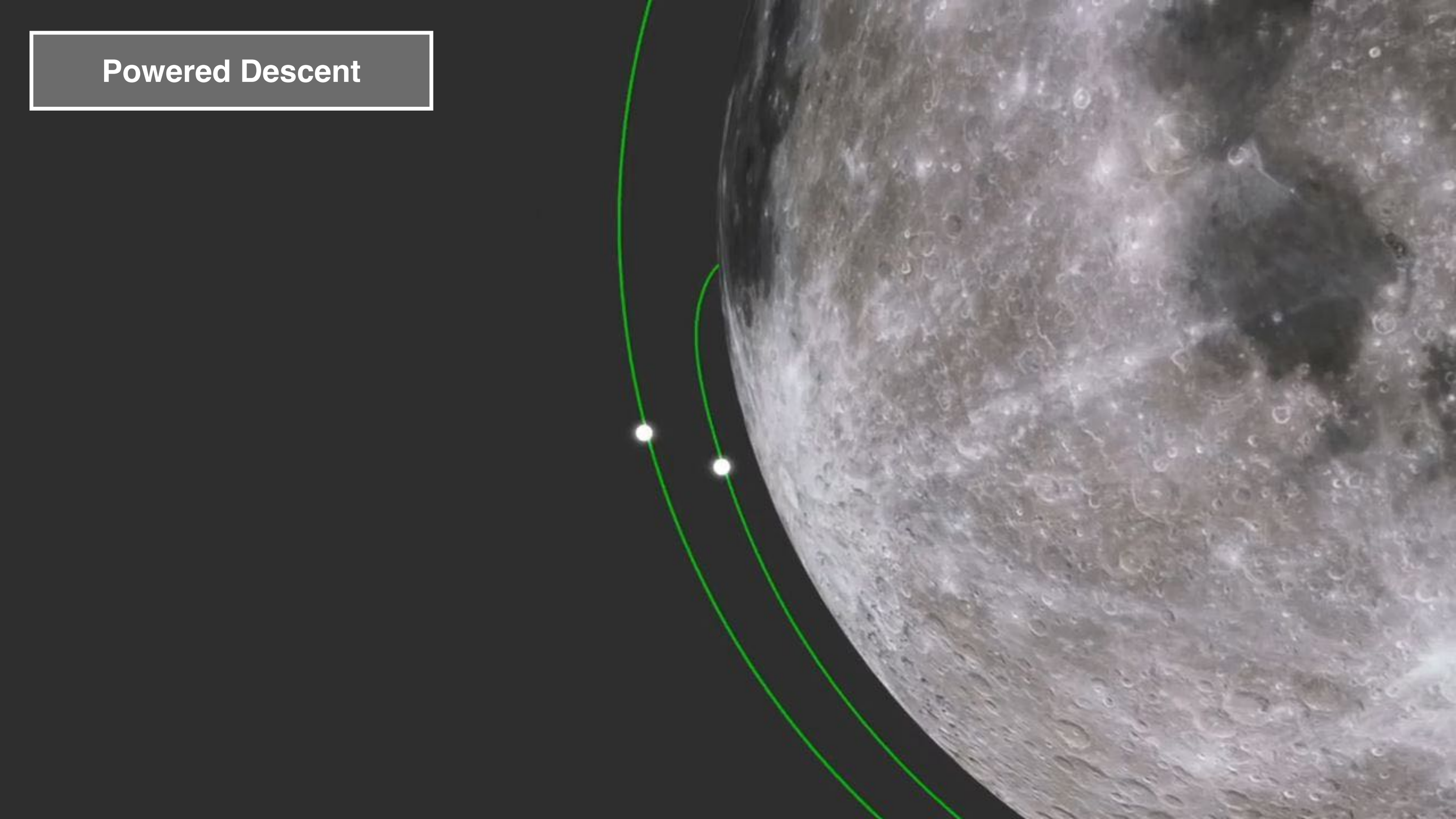
Lunar Orbit Insertion

youtube.com/JaredOwen

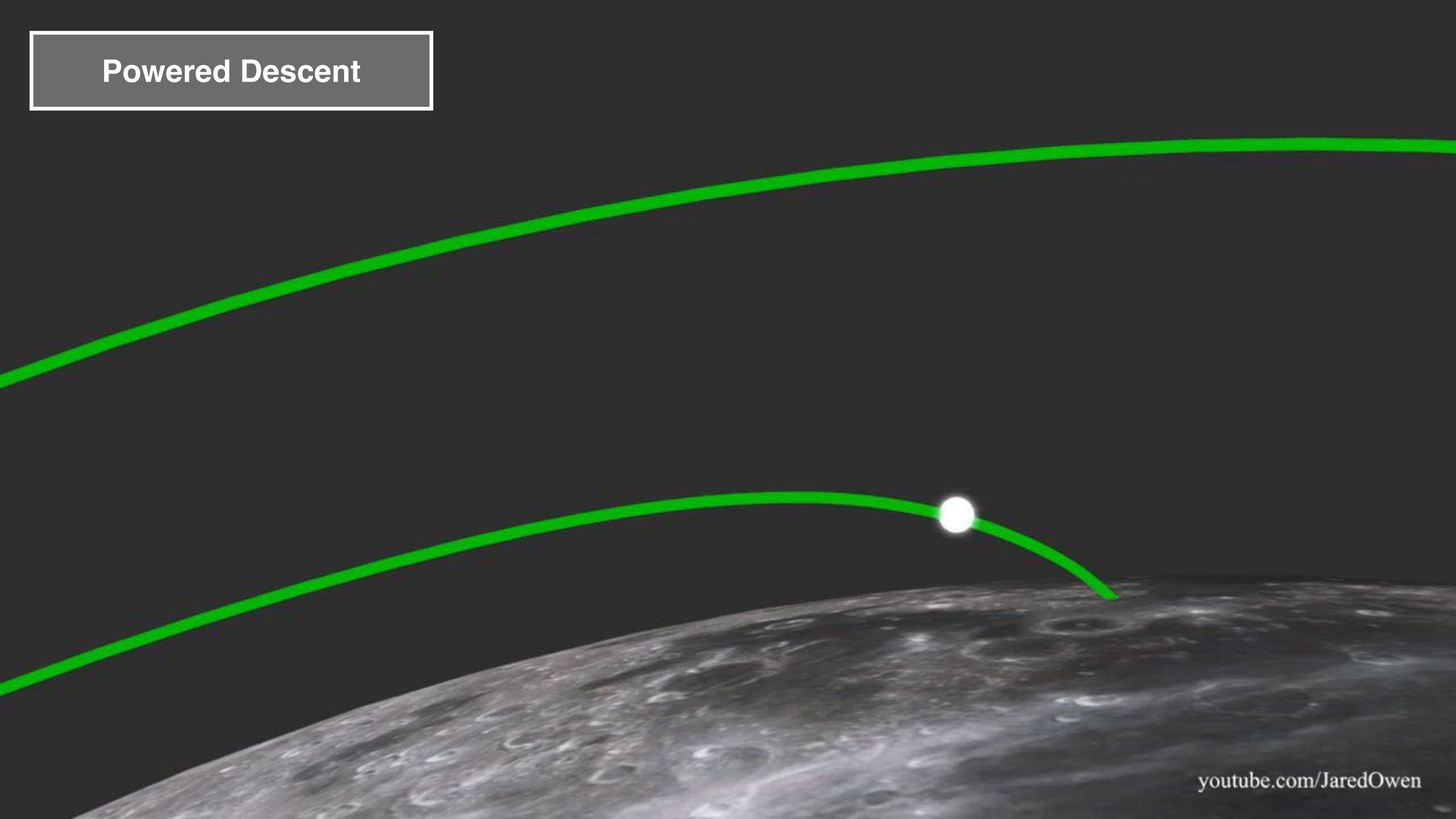
Undocking

Undocking

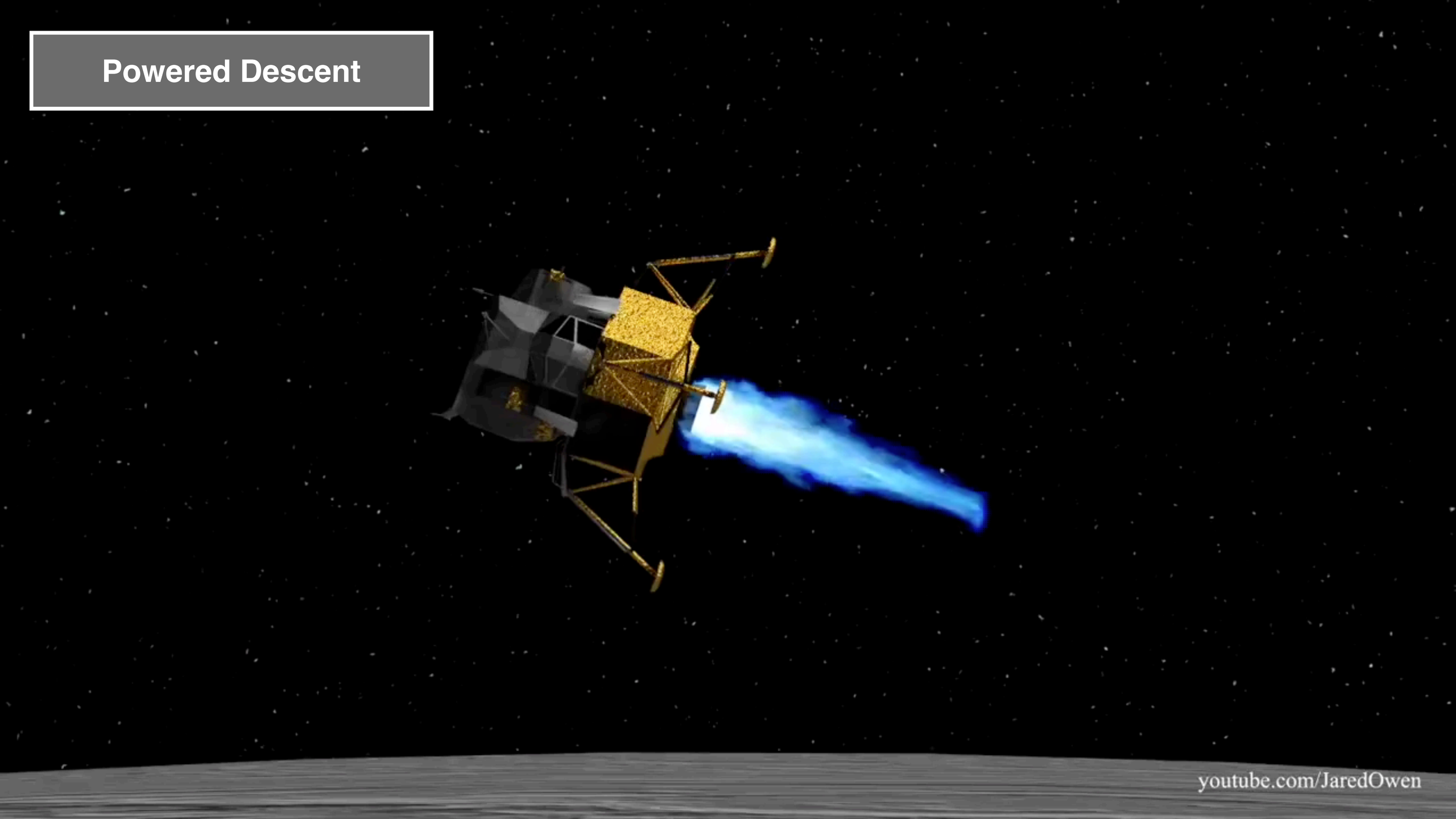
Powered Descent



Powered Descent

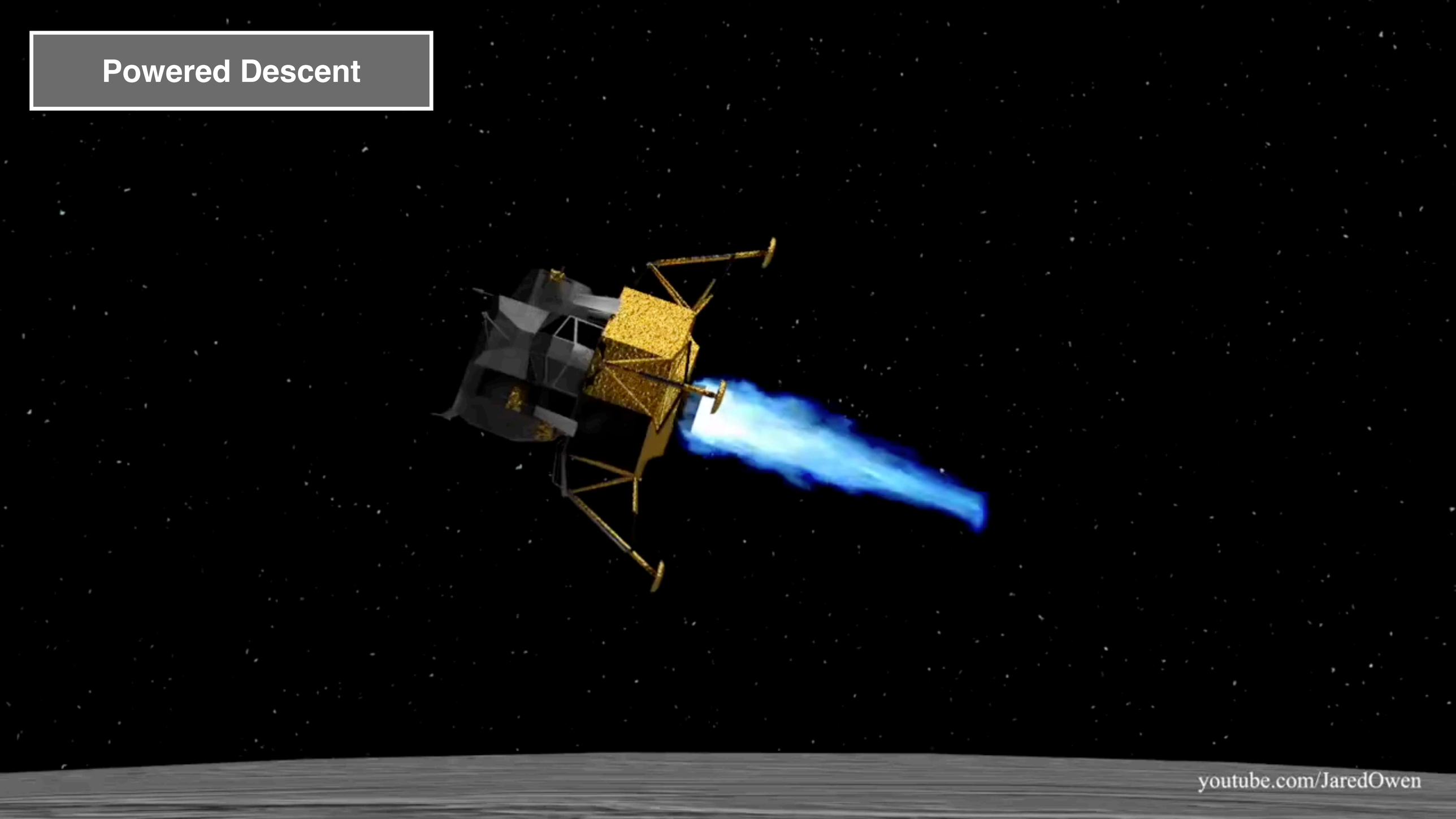


Powered Descent



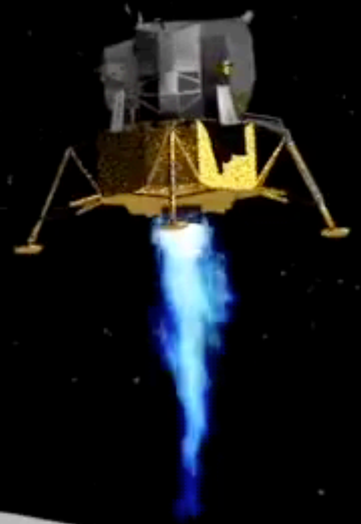
youtube.com/JaredOwen

Powered Descent



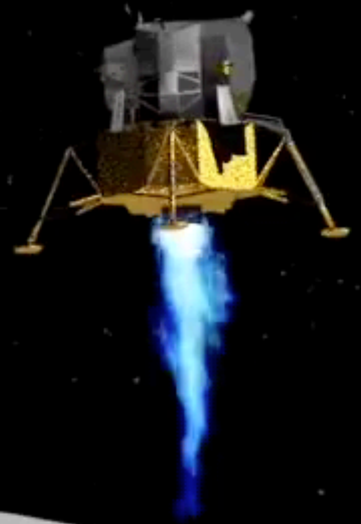
youtube.com/JaredOwen

Landing



youtube.com/JaredOwen

Landing



youtube.com/JaredOwen

Ascent

Ascent

Ascent

Lunar Orbit Rendezvous

youtube.com/JaredOwen

Lunar Orbit Rendezvous

youtube.com/JaredOwen

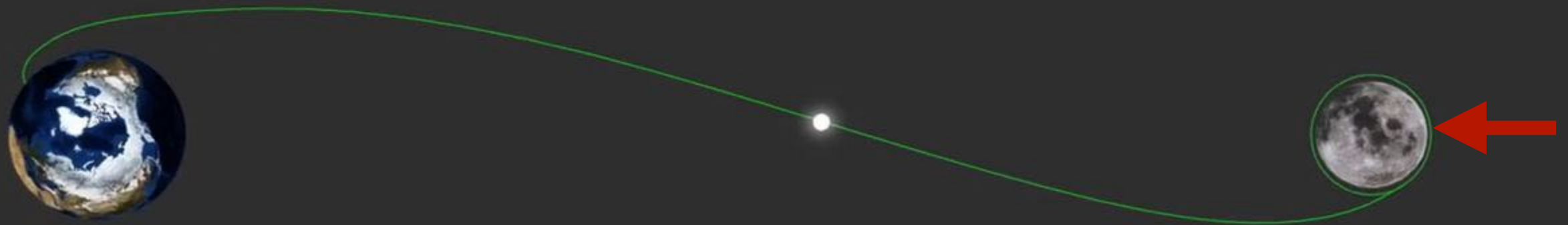
Lunar Orbit Rendezvous

youtube.com/JaredOwen

Trans-Earth Injection

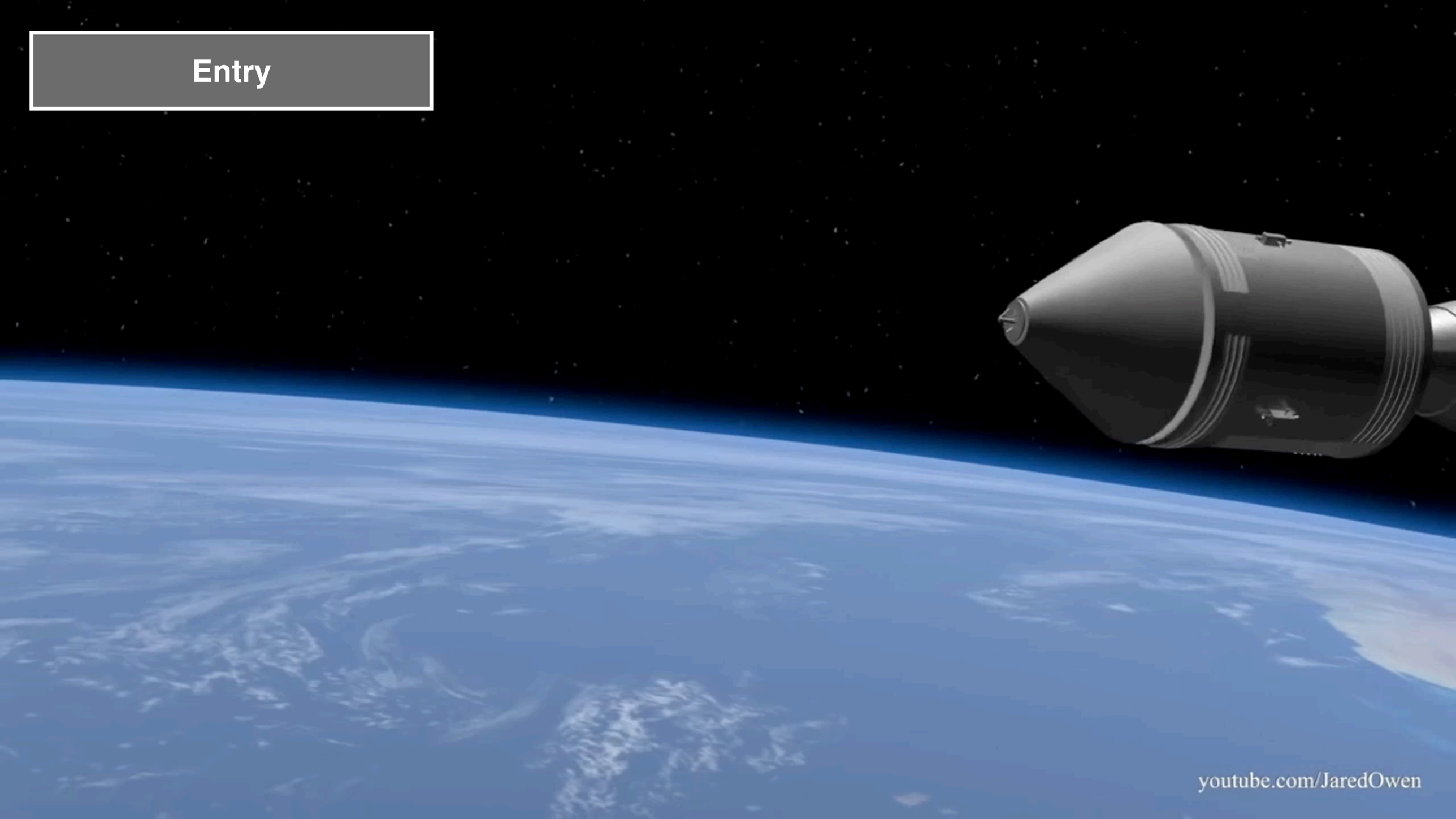
youtube.com/JaredOwen

Trans-Earth Injection



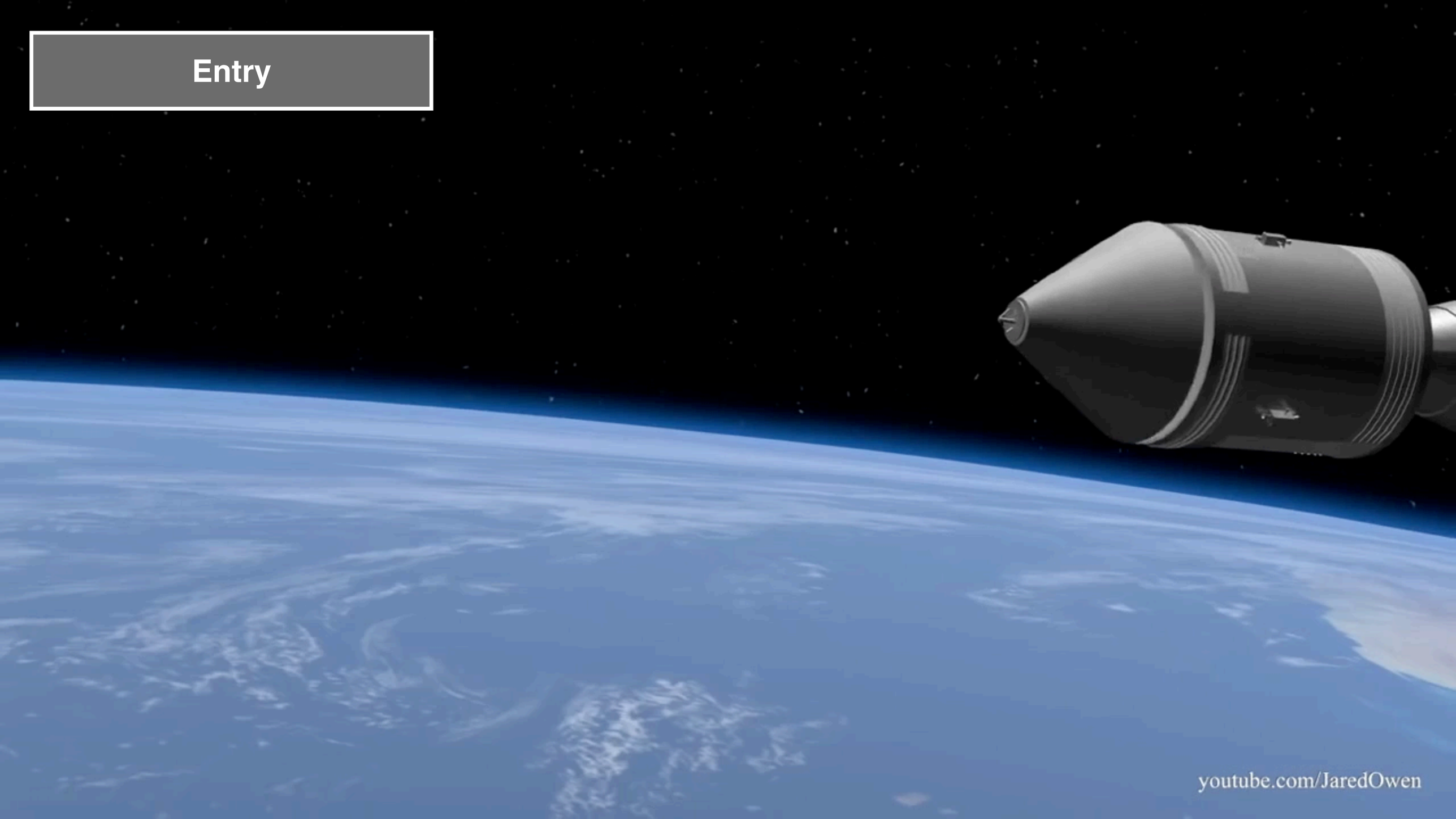
youtube.com/JaredOwen

Entry

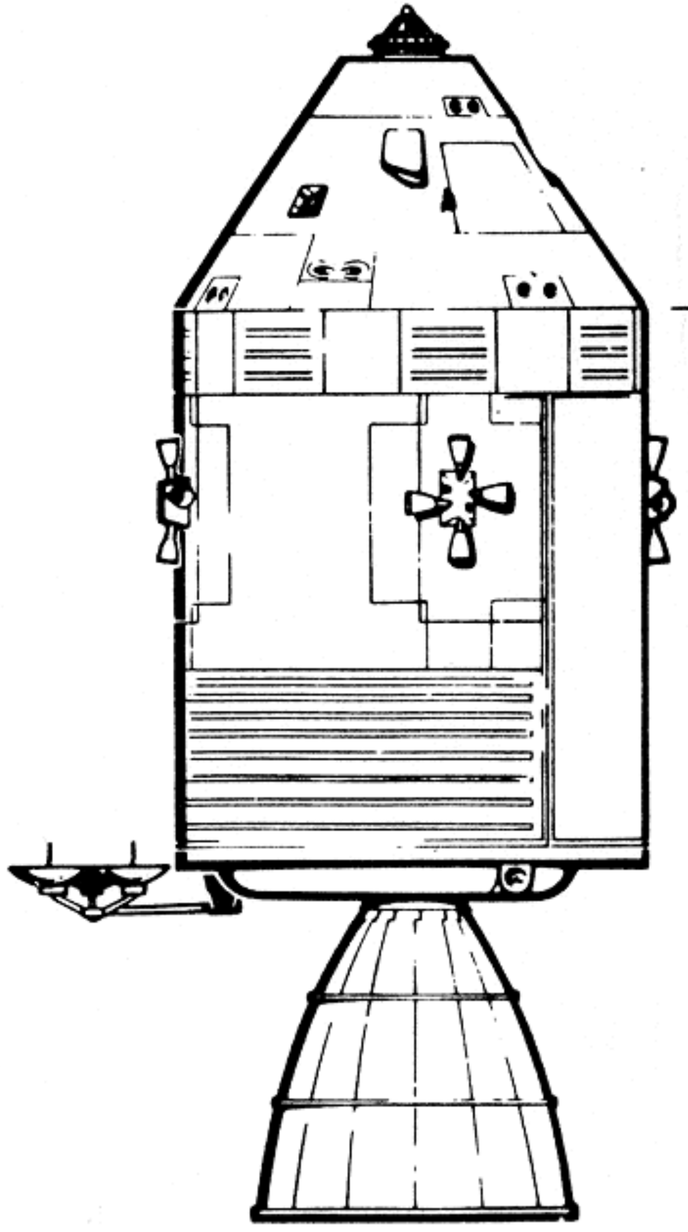


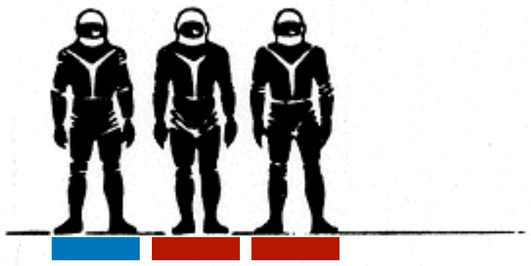
youtube.com/JaredOwen

Entry

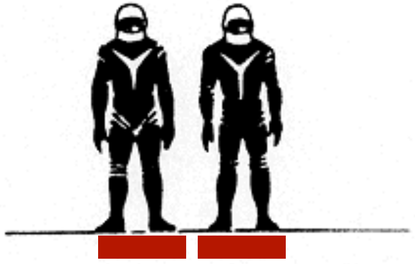


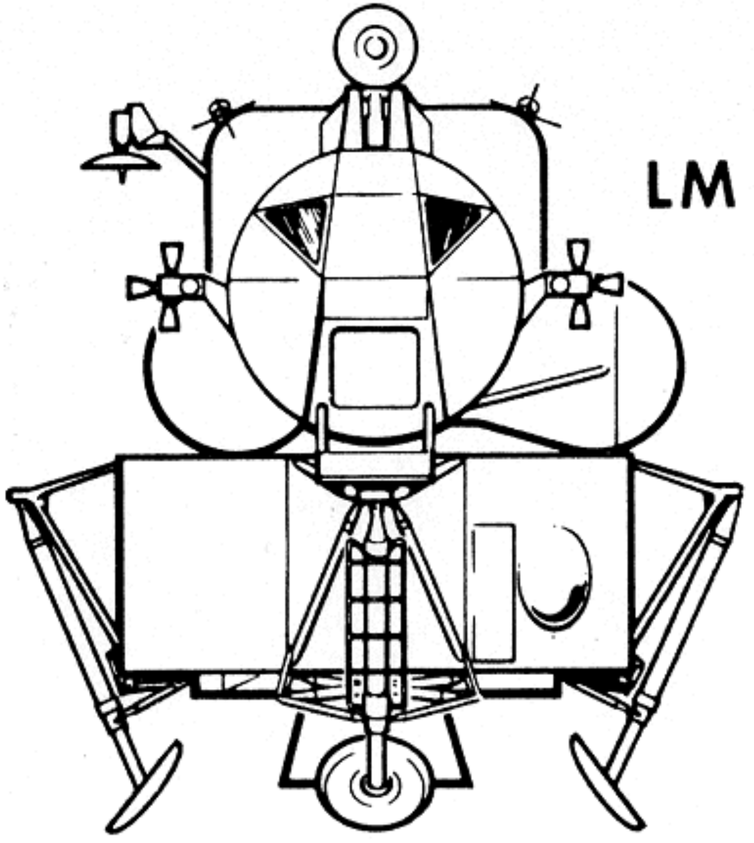
youtube.com/JaredOwen



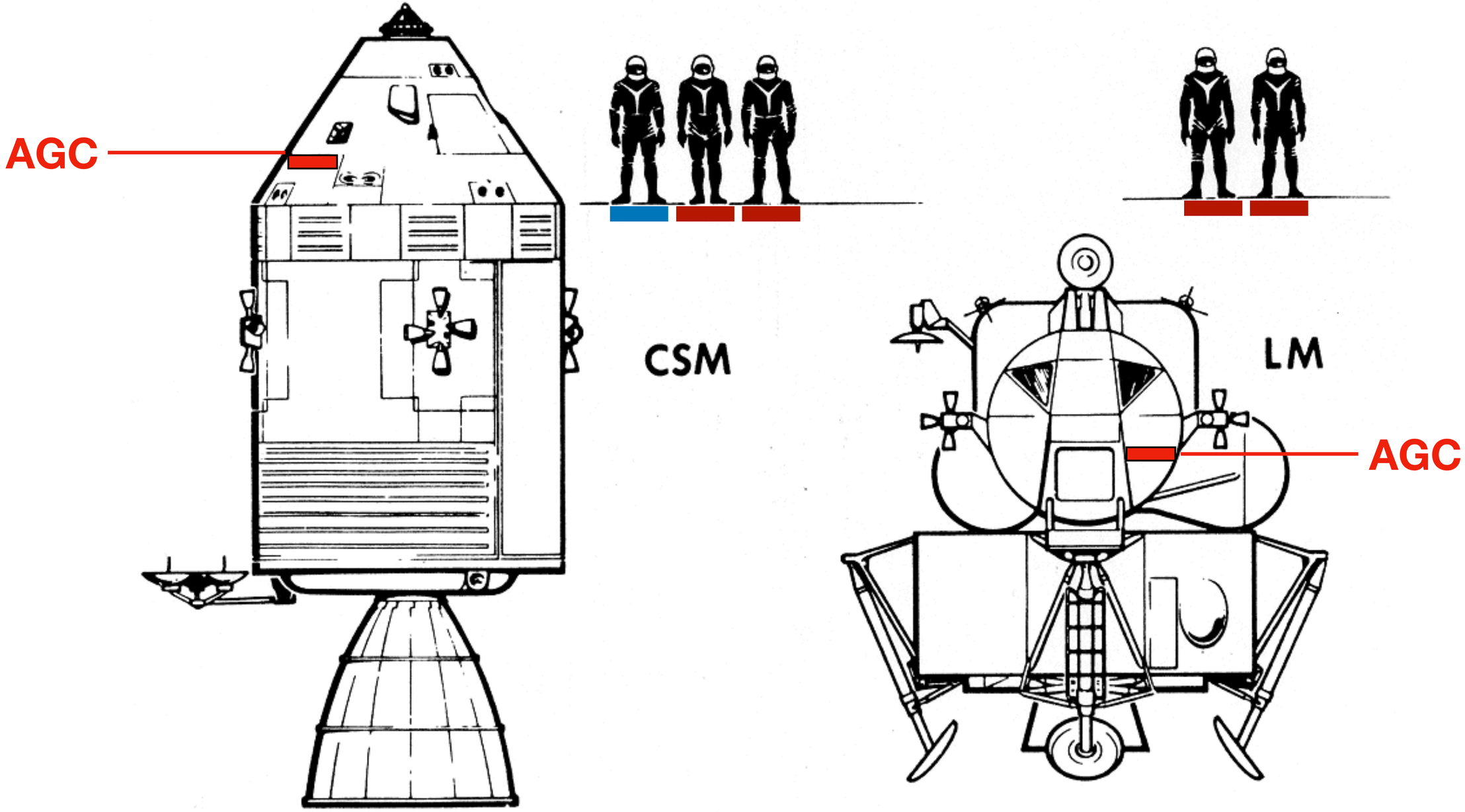


CSM

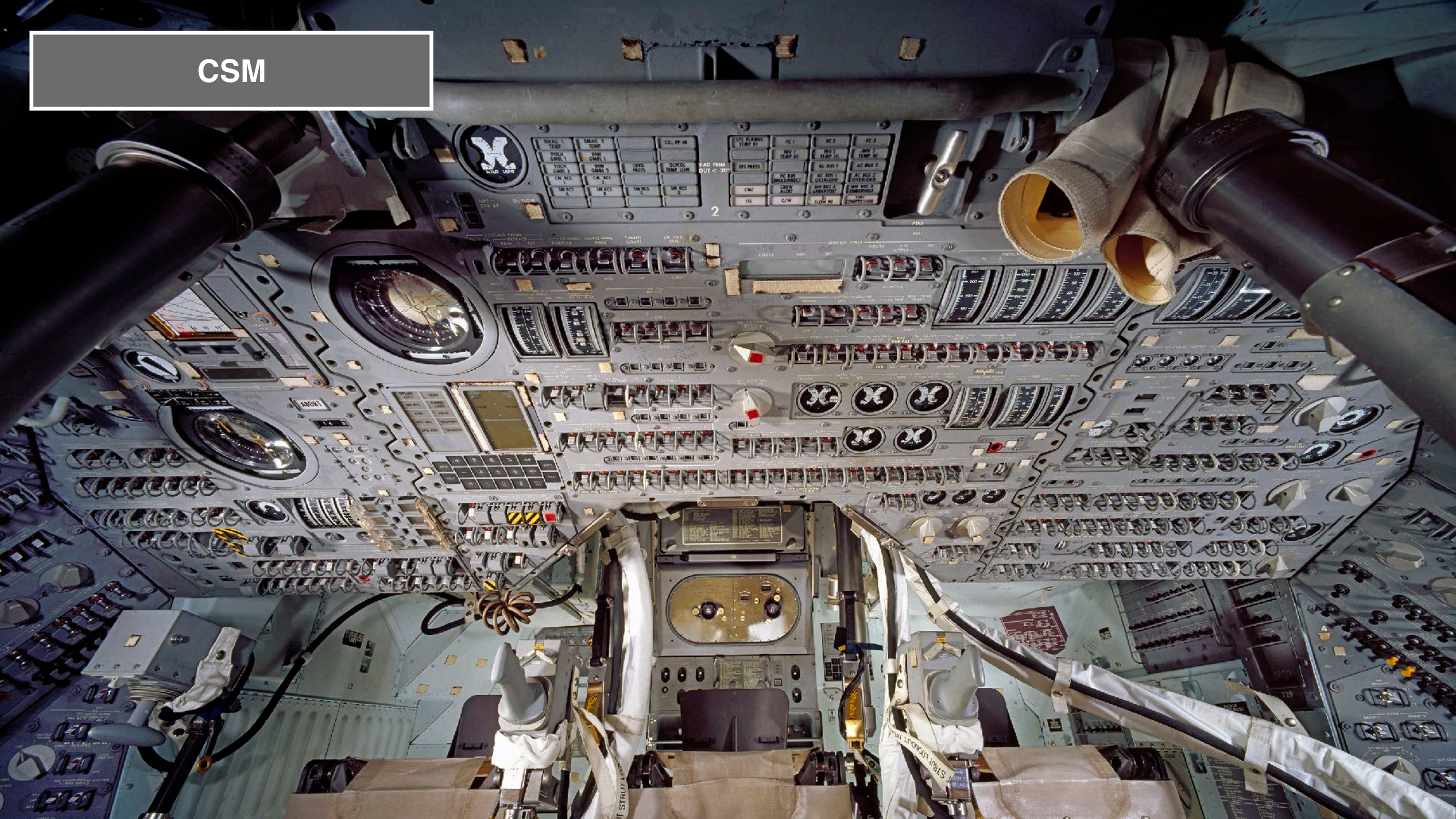




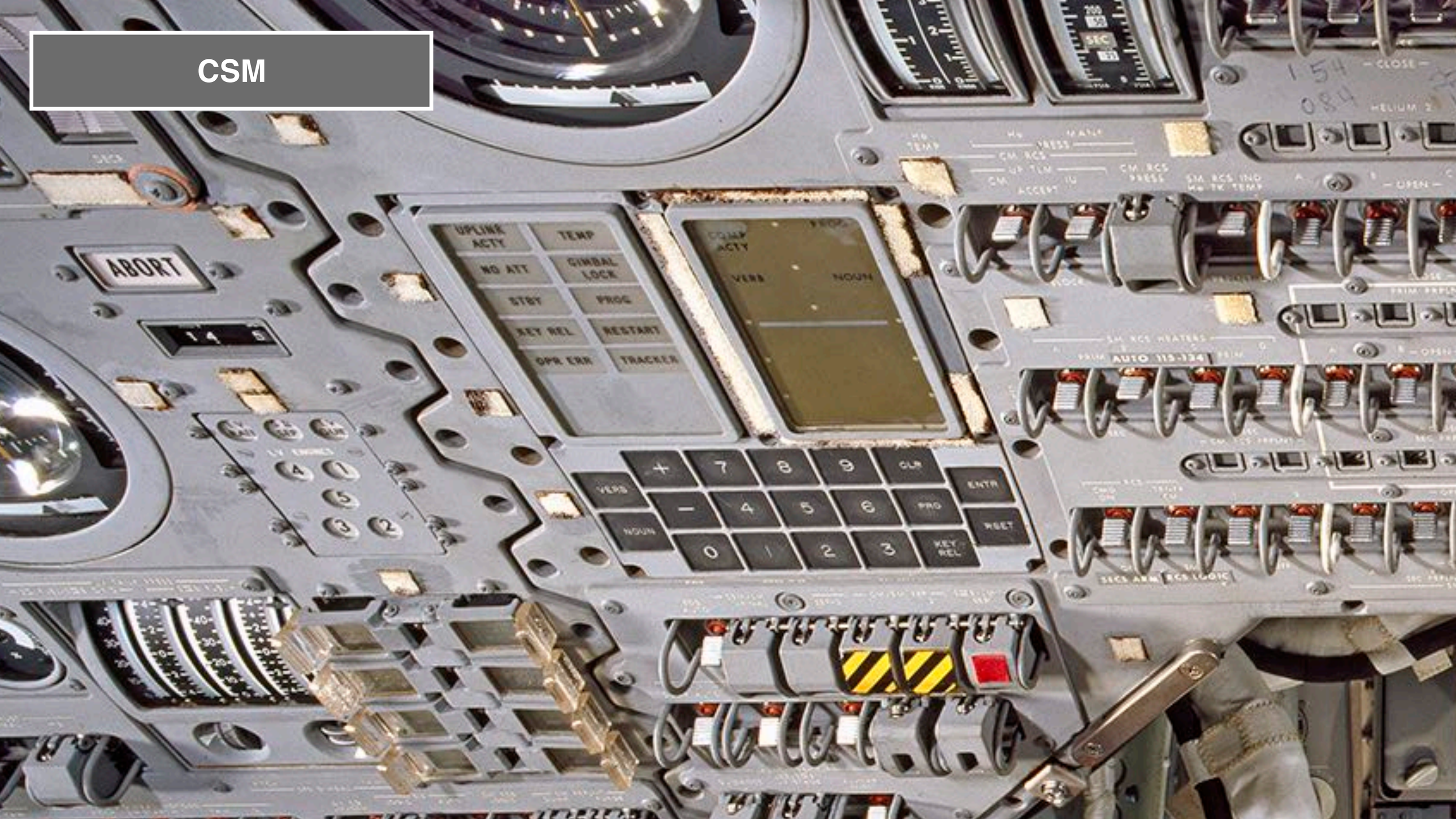
LM


AGC
CSM
LM
AGC

CSM



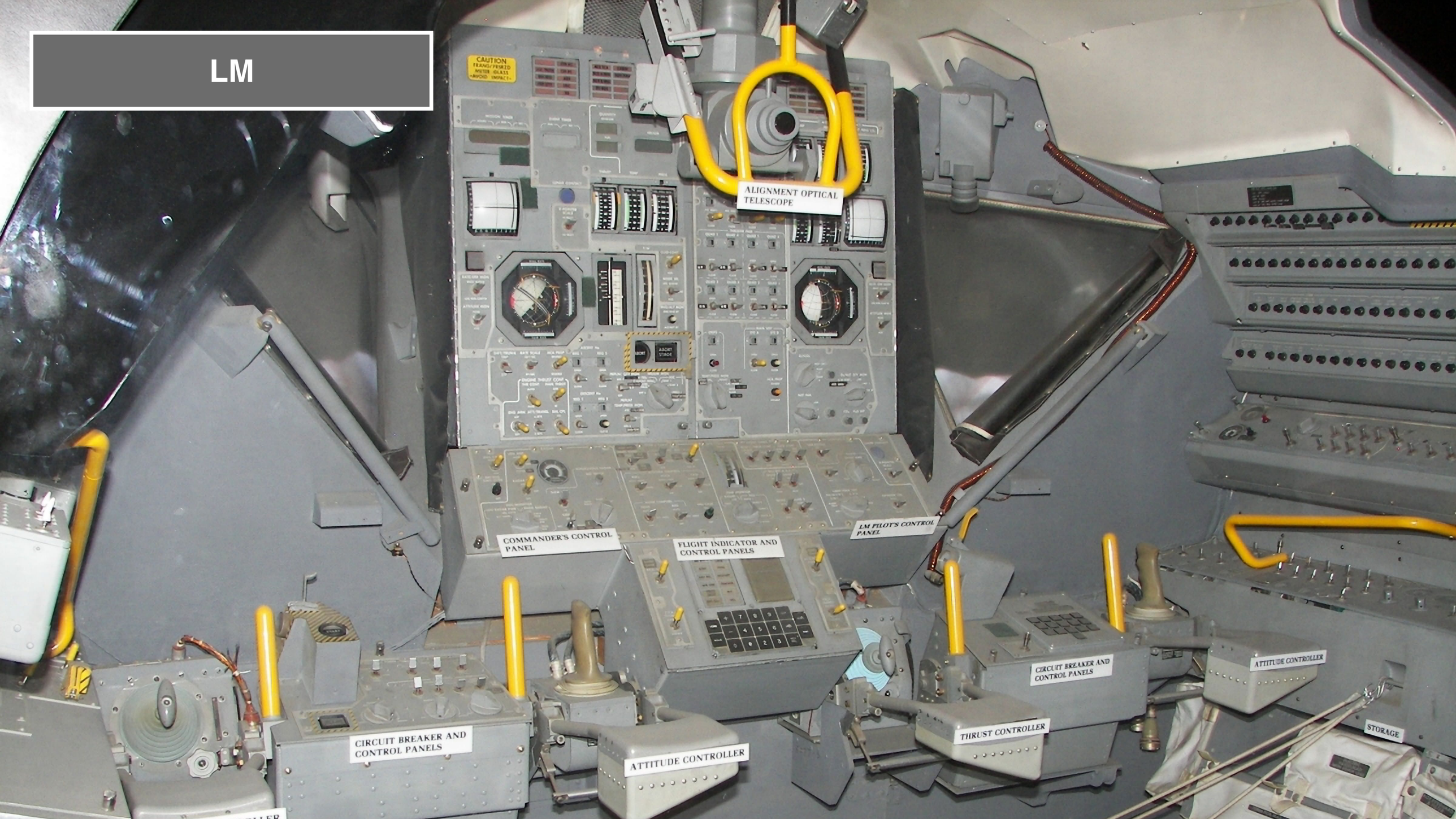
CSM


ABORT
UPLINK ACTY
TEMP
NO ATT
GIMBAL LOCK
STBY
PROG
KEY REL
RESTART
OPR ERR
TRACKER
VERB
NOUN
CLR
ENTR
RSET
PRO
KEY REL
AUTO 115-124

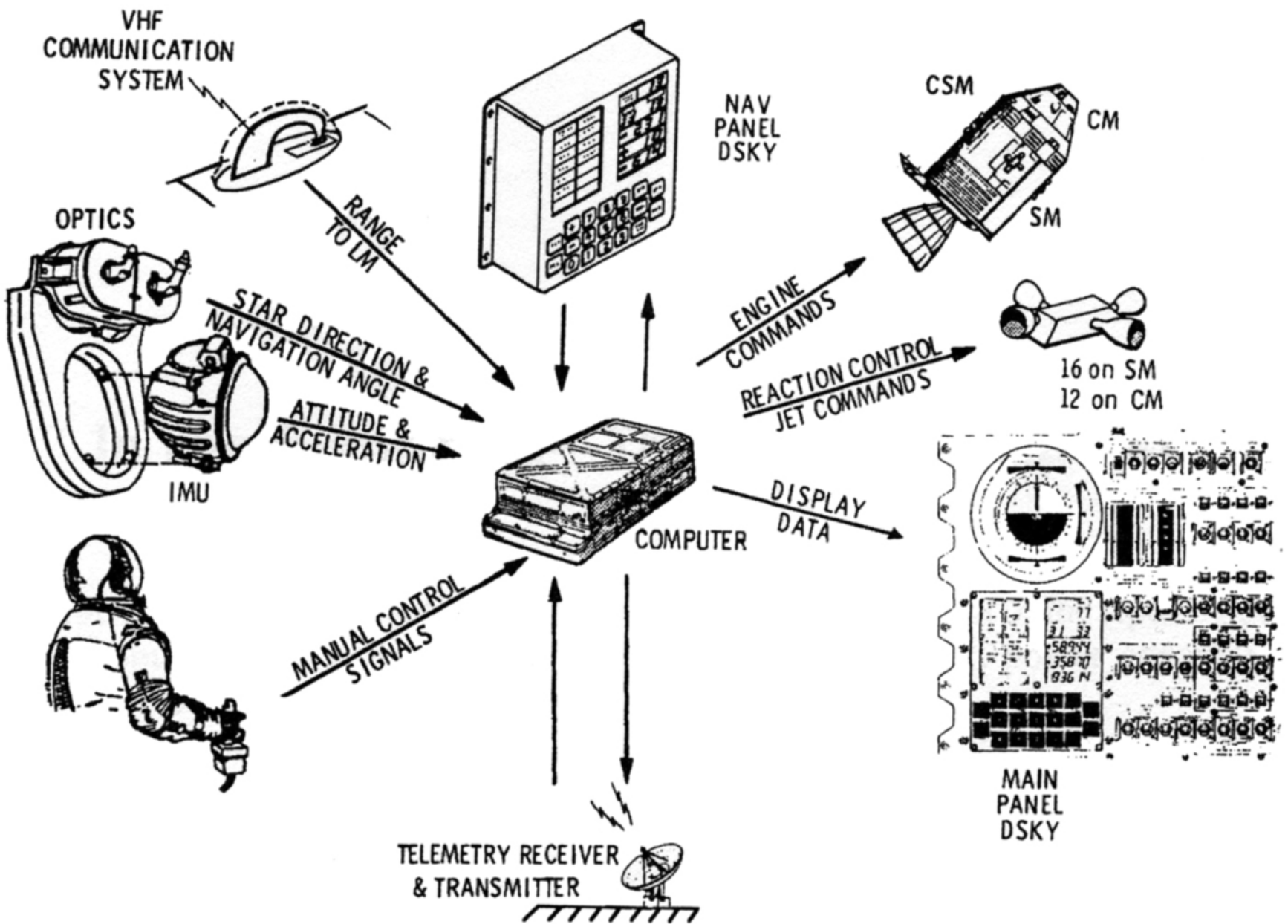
LM


COMMANDER'S CONTROL PANEL
FLIGHT INDICATOR AND CONTROL PANELS
LM PILOT'S CONTROL PANEL

LM

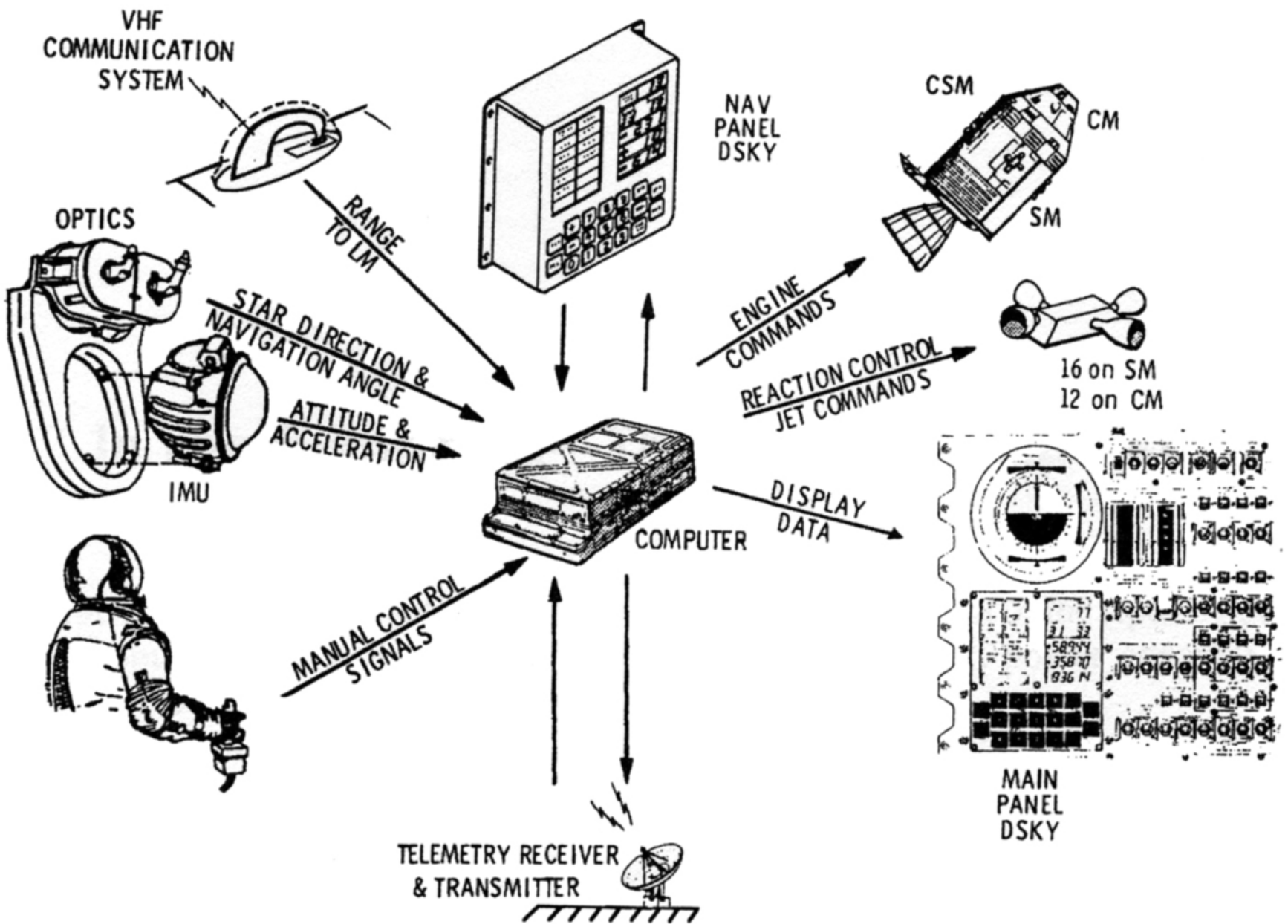

ALIGNMENT OPTICAL TELESCOPE
COMMANDER'S CONTROL PANEL
FLIGHT INDICATOR AND CONTROL PANELS
LM PILOT'S CONTROL PANEL
CIRCUIT BREAKER AND CONTROL PANELS
ATTITUDE CONTROLLER
THRUST CONTROLLER
CIRCUIT BREAKER AND CONTROL PANELS
ATTITUDE CONTROLLER
STORAGE

AGC Responsibilities

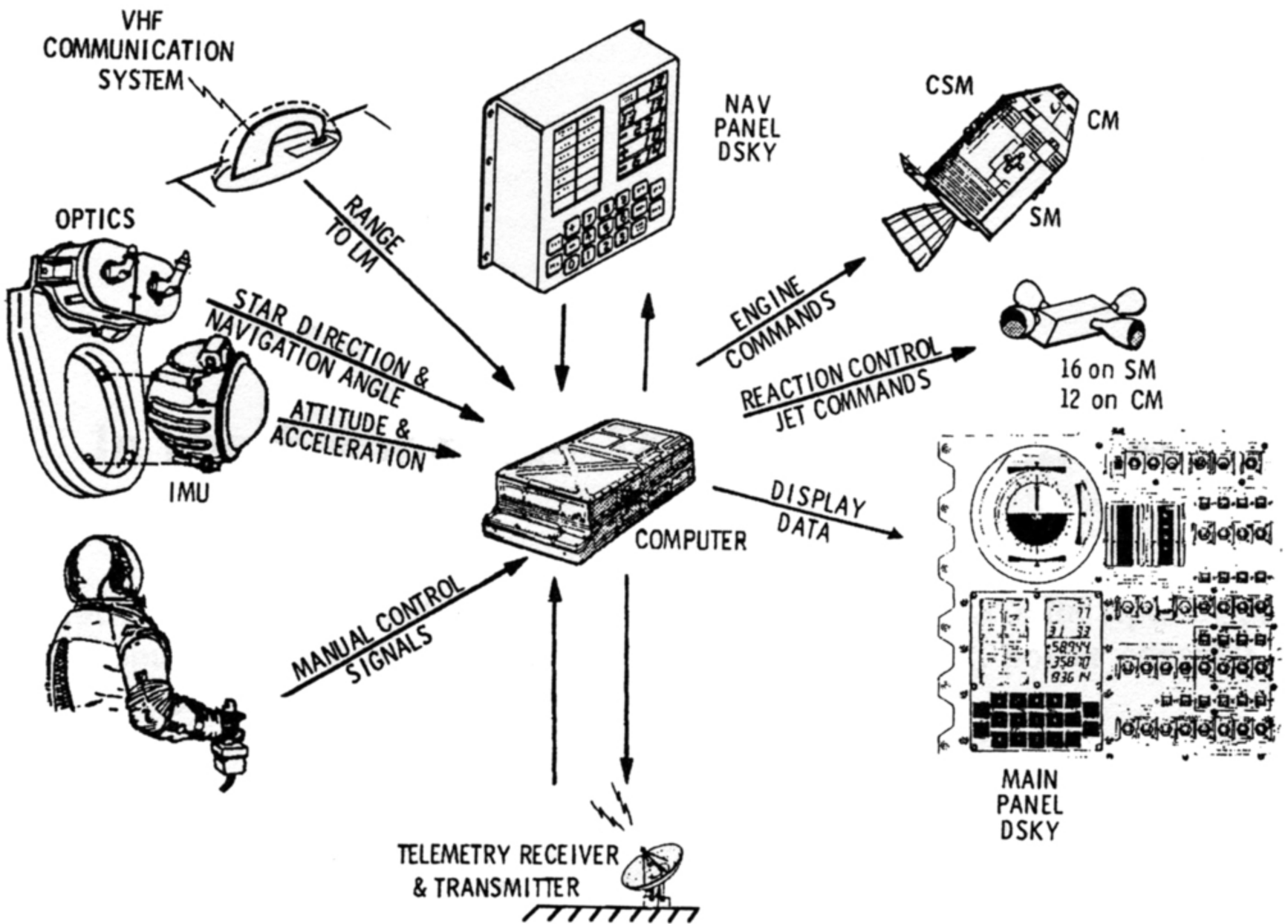


AGC Responsibilities

Maintain State Vector



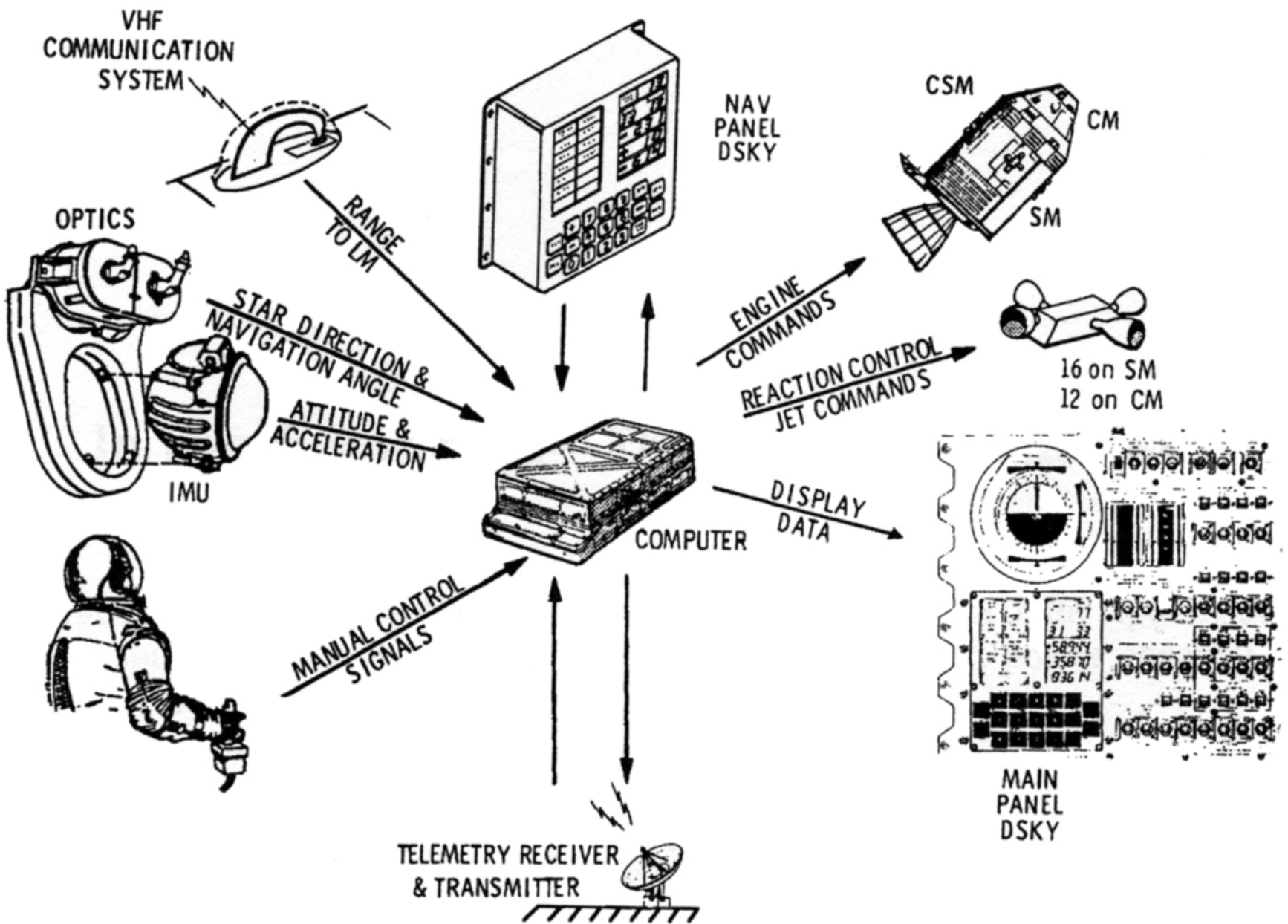
AGC Responsibilities



Maintain State Vector

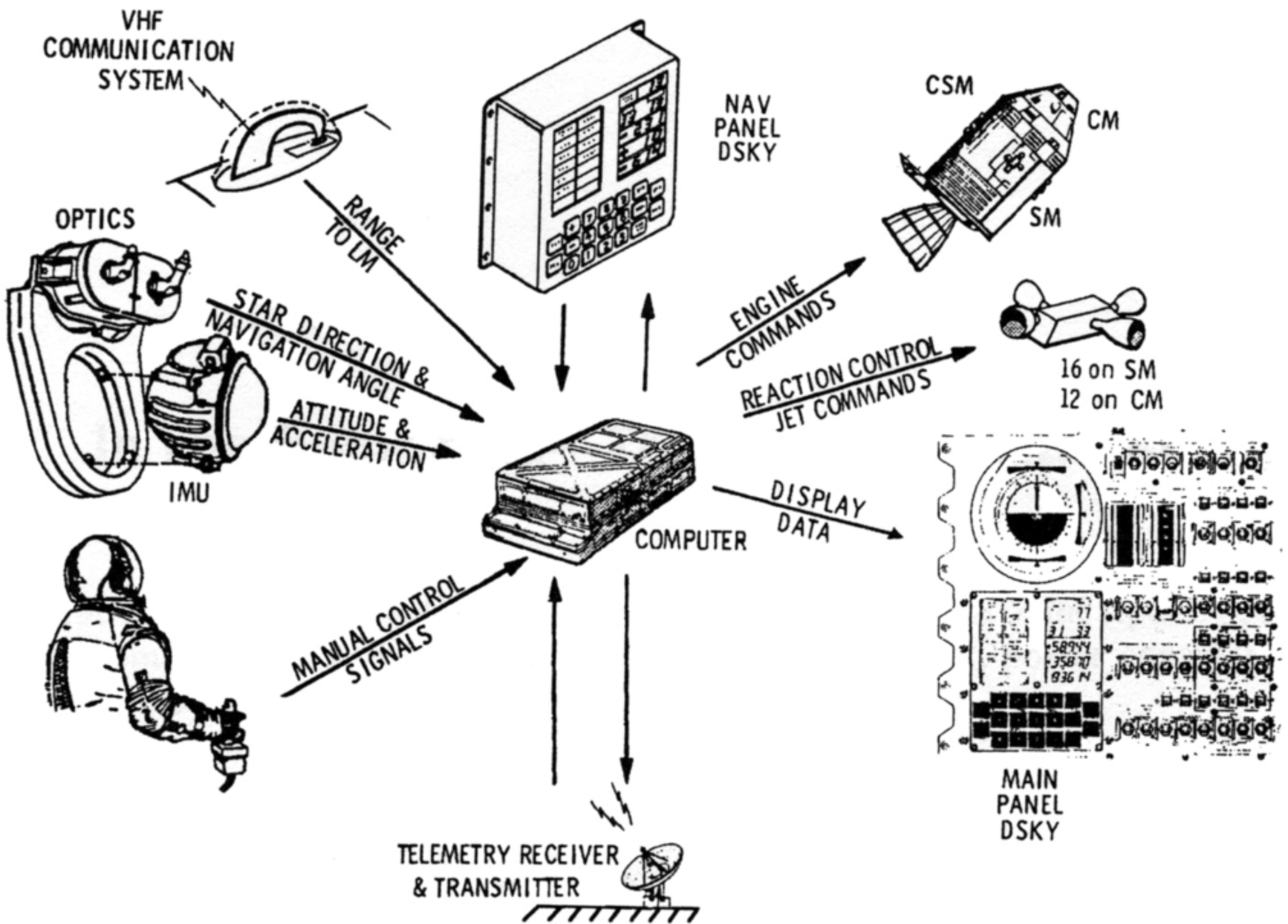
Stabilize Attitude

AGC Responsibilities



- Maintain State Vector
- Stabilize Attitude
- Calculate Burns
- Control Burns

AGC Responsibilities



- Maintain State Vector
- Stabilize Attitude
- Calculate Burns
- Control Burns
- Monitor/Control Launch

Maintain State Vector

Calculate Burns

Stabilize Attitude

Control Burns

Monitor/Control Launch

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

Architecture

Architecture

Architecture

- von Neumann
- Accumulator Machine
- 15 bit
- One’s Complement
- Big Endian

Architecture

Architecture

- Instruction Set
- Arithmetic
- Encoding

Architecture

- Instruction Set
- Memory Model
- Arithmetic
- I/O
- Encoding
- Counters

Architecture

- Instruction Set
- Memory Model
- Arithmetic
- I/O
- Interrupts
- Encoding
- Counters

armv8

abs
adc
adcs
add
addhn
addhn2
addp
adds
addv
adr
adrp
aesd
aese
aesimc
aesmc
and
ands
asrv
b
b.c
bfm
bic
bics
bif
bit
bl
blr
br
brk
bsl
cbnz
cbz
ccmn
ccmp
clrex
cls
clz
cmeq
cmge
cmgt
cmhi
cmhs
cmle
cmlt
cmtst
cnt
csel
csinc
csinv
csneg
dcps1
dcps2
dcps3
dmb
drps
dsb
dup
eon
eor
eret
ext
extr
fabd
fabs
facge
facgt
fadd
faddp
fccmp
fccmpe
fcmeq
fcmge
fcmgt
fcmle
fcmlt
fcmp
fcmpe
fcsel
fcvt
fcvtas
fcvtau
fcvtl
fcvtl2
fcvtms
fcvtmu
fcvtn
fcvtn2
fcvtns
fcvtnu
fcvtps
fcvtpu
fcvtxn
fcvtxn2
fcvtzs
fcvtzu
fdiv
fmadd
fmax
fmaxnm
fmaxnmp
fmaxnmv
fmaxp
fmaxv
fmin
fminnm
fminnmp
fminnmv
fminp
fminv
fmla
fmls
fmov
fmsub
fmul
fmulx
fneg
fnmadd
fnmsub
fnmul
frecpe
frecps
frecpx
frinta
frinti
frintm
frintn
frintp
frintx
frintz
frsqrte
frsqrts
fsqrt
fsub
hint
hlt
hvc
ins
isb
ld1
ld2
ld3
ld4
ldar
ldarb
ldarh
ldaxp
ldaxr
ldaxrb
ldaxrh
ldnp
ldp
ldpsw
ldr
ldrb
ldrh
ldrsb
ldrsh
ldrsw
ldtr
ldtrb
ldtrh
ldtrsb
ldtrsh
ldtrsw
ldur
ldurb
ldurh
ldursb
ldursh
ldursw
ldxp
ldxr
ldxrb
ldxrh
lslv
lsrv
madd
mla
mls
movi
movk
movn
movz
mrs
msr
msub
mul
mvni
neg
not
orn
orr
pmul
pmull
pmull2
prfm
prfum
raddhn
raddhn2
rbit
ret
rev
rev16
rev32
rev64
rorv
rshrn
rshrn2
rsubhn
rsubhn2
saba
sabal
sabal2
sabd
sabdl
sabdl2
sadalp
saddl
saddl2
saddlp
saddlv
saddw
saddw2
sbc
sbcs
sbfm
scvtf
sdiv
sha1c
sha1h
sha1m
sha1p
sha1su0
sha1su1
sha256h
sha256h2
sha256su0
sha256su1
shadd
shl
shll
shll2
shsub
sli
smaddl
smax
smaxp
smaxv
smc
smin
sminp
sminv
smlal
smlal2
smlsl
smlsl2
smov
smsubl
smulh
smull
smull2
sqabs
sqadd
sqdmlal
sqdmlal2
sqdmlsl
sqdmlsl2
sqdmulh
sqdmull
sqdmull2
sqneg
sqrdmulh
sqrshl
sqrshrn
sqrshrn2
sqrshrun
sqrshrun2
sqshl
sqshlu
sqshrn
sqshrun
sqsub
sqxtn
sqxtn2
sqxtun
sqxtun2
srhadd
sri
srshl
srshr
srsra
sshl
sshr
ssra
ssubl
ssubl2
ssubw
ssubw2
st1
st2
st3
st4
stlr
stlrb
stlrh
stlxp
stlxr
stlxrb
stlxrh
stnp
stp
str
strb
strh
sttr
sttrb
sttrh
stur
sturb
sturh
stxp
stxr
stxrb
stxrh
sub
subhn
subhn2
subs
suqadd
svc
sys
sysl
tbl
tbnz
tbx
tbz
trn1
trn2
uaba
uabal
uabal2
uabd
uabdl
uabdl2
uadalp
uaddl
uaddl2
uaddlp
uaddlv
uaddw
uaddw2
ubfm
ucvtf
udiv
uhadd
uhsub
umaddl
umax
umaxp
umaxv
umin
uminp
uminv
umlal
umlal2
umlsl
umlsl2
umov
umsubl
umulh
umull
umull2
uqadd
uqrshl
uqrshrn
uqrshrn2
uqshl
uqshrn
uqsub
uqxtn
uqxtn2
urecpe
urhadd
urshl
urshr
ursqrte
ursra
ushl
ushr
usqadd
usra
usubl
usubl2
usubw
usubw2
uzp1
uzp2
xtn
xtn2
zip1
zip2

http://kitoslab-eng.blogspot.com/2012/10/armv8-aarch64-instruction-encoding.html

subleq

```
subleq a, b, c
```

armv8

abs	cmhs	fcvtl2	frinti	ldur	rorv	smaxp	sri	suqadd	umulh
adc	cmle	fcvtms	frintm	ldurb	rshrn	smaxv	srshl	svc	umull
adcs	cmlt	fcvtmu	frintn	ldurh	rshrn2	smc	srshr	sys	umull2
add	cmtst	fcvtn	frintp	ldursb	rsubhn	smin	srsra	sysl	uqadd
addhn	cnt	fcvtn2	frintx	ldursh	rsubhn2	sminp	sshl	tbl	uqrshl
addhn2	csel	fcvtns	frintz	ldursw	saba	sminv	sshr	tbnz	uqrshrn
addp	csinc	fcvtnu	frsqrte	ldxp	sabal	smlal	ssra	tbx	uqrshrn2
adds	csinv	fcvtps	frsqrts	ldxr	sabal2	smlal2	ssubl	tbz	uqshl
addv	csneg	fcvtpu	fsqrt	ldxrb	sabd	smlsl	ssubl2	trn1	uqshrn
adr	dcps1	fcvtxn	fsub	ldxrh	sabdl	smlsl2	ssubw	trn2	uqsub
adrp	dcps2	fcvtxn2	hint	lslv	sabdl2	smov	ssubw2	uaba	uqxtn
aesd	dcps3	fcvtzs	hlt	lsrv	sadalp	smsubl	st1	uabal	uqxtn2
aese	dmb	fcvtzu	hvc	madd	saddl	smulh	st2	uabal2	urecpe
aesimc	drps	fdiv	ins	mla	saddl2	smull	st3	uabd	urhadd
aesmc	dsb	fmadd	isb	mls	saddlp	smull2	st4	uabdl	urshl
and	dup	fmax	ld1	movi	saddlv	sqabs	stlr	uabdl2	urshr
ands	eon	fmaxnm	ld2	movk	saddw	sqadd	stlrb	uadalp	ursqrte
asrv	eor	fmaxnmp	ld3	movn	saddw2	sqdmlal	stlrh	uaddl	ursra
b	eret	fmaxnmv	ld4	movz	sbc	sqdmlal2	stlxp	uaddl2	ushl
b.c	ext	fmaxp	ldar	mrs	sbcs	sqdmlsl	stlxr	uaddlp	ushr
bfm	extr	fmaxv	ldarb	msr	sbfm	sqdmlsl2	stlxrb	uaddlv	usqadd
bic	fabd	fmin	ldarh	msub	scvtf	sqdmulh	stlxrh	uaddw	usra
bics	fabs	fminnm	ldaxp	mul	sdiv	sqdmull	stnp	uaddw2	usubl
bif	facge	fminnmp	ldaxr	mvni	sha1c	sqdmull2	stp	ubfm	usubl2
bit	facgt	fminnmv	ldaxrb	neg	sha1h	sqneg	str	ucvtf	usubw
bl	fadd	fminp	ldaxrh	not	sha1m	sqrdmulh	strb	udiv	usubw2
blr	faddp	fminv	ldnp	orn	sha1p	sqrshl	strh	uhadd	uzp1
br	fccmp	fmla	ldp	orr	sha1su0	sqrshrn	sttr	uhsub	uzp2
brk	fccmpe	fmls	ldpsw	pmul	sha1su1	sqrshrn2	sttrb	umaddl	xtn
bsl	fcmeq	fmov	ldr	pmull	sha256h	sqrshrun	sttrh	umax	xtn2
cbnz	fcmge	fmsub	ldrb	pmull2	sha256h2	sqrshrun2	stur	umaxp	zip1
cbz	fcmgt	fmul	ldrh	prfm	sha256su0	sqshl	sturb	umaxv	zip2
ccmn	fcmle	fmulx	ldrsb	prfum	sha256su1	sqshlu	sturh	umin	
ccmp	fcmlt	fneg	ldrsh	raddhn	shadd	sqshrn	stxp	uminp	
clrex	fcmp	fnmadd	ldrsw	raddhn2	shl	sqshrun	stxr	uminv	
cls	fcmpe	fnmsub	ldtr	rbit	shll	sqsub	stxrb	umlal	
clz	fcsel	fnmul	ldtrb	ret	shll2	sqxtn	stxrh	umlal2	
cmeq	fcvt	frecpe	ldtrh	rev	shsub	sqxtn2	sub	umlsl	
cmge	fcvtas	frecps	ldtrsb	rev16	sli	sqxtun	subhn	umlsl2	
cmgt	fcvtau	frecpx	ldtrsh	rev32	smaddl	sqxtun2	subhn2	umov	
cmhi	fcvtl	frinta	ldtrsw	rev64	smax	srhadd	subs	umsubl	

http://kitoslab-eng.blogspot.com/2012/10/armv8-aarch64-instruction-encoding.html

subleq

```
subleq a, b, c
```

```
M[b] ← M[b] - M[a];
if M[b] ≤ 0 goto c;
```

armv8

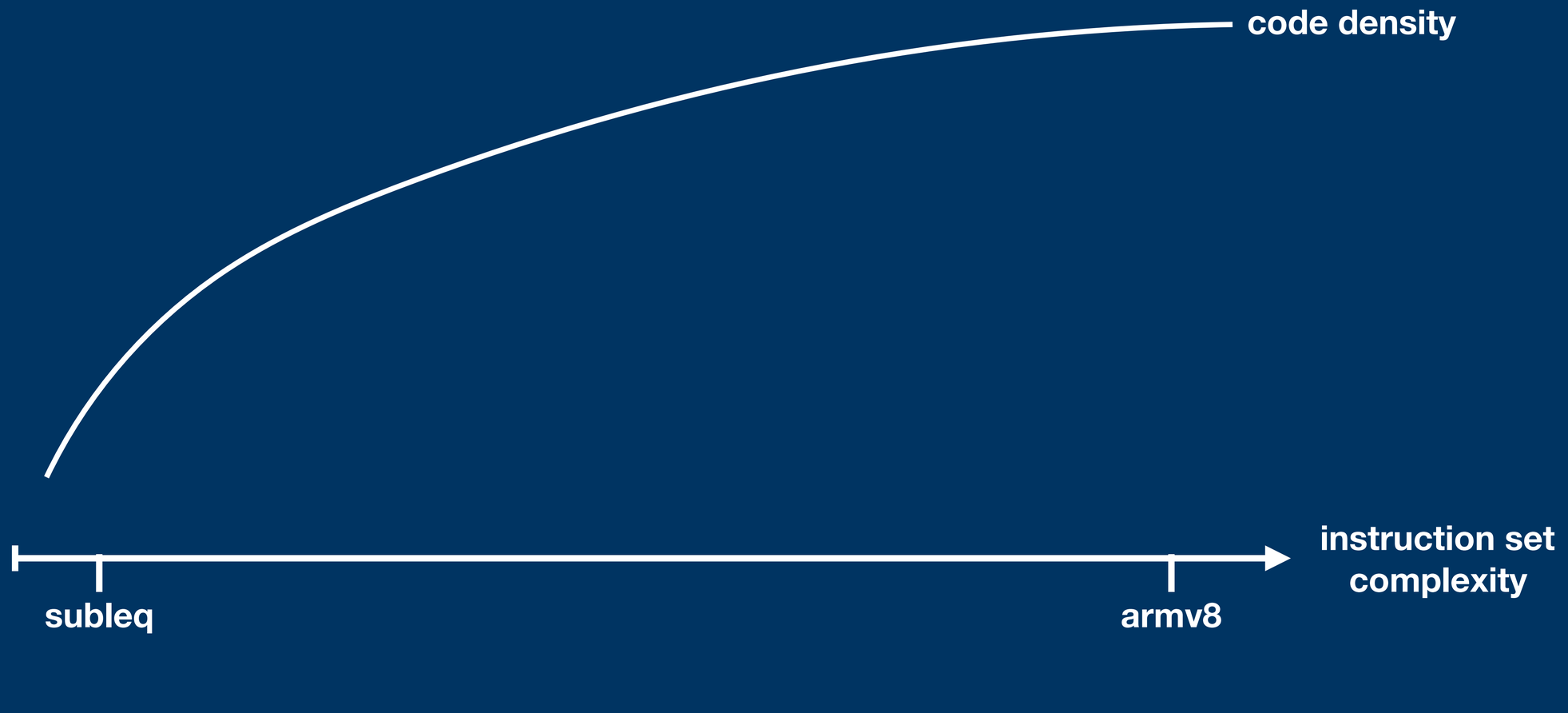
abs	cmhs	fcvtl2	frinti	ldur	rorv	smaxp	sri	suqadd	umulh
adc	cmle	fcvtms	frintm	ldurb	rshrn	smaxv	srshl	svc	umull
adcs	cmlt	fcvtmu	frintn	ldurh	rshrn2	smc	srshr	sys	umull2
add	cmtst	fcvtn	frintp	ldursb	rsubhn	smin	srsra	sysl	uqadd
addhn	cnt	fcvtn2	frintx	ldursh	rsubhn2	sminp	sshl	tbl	uqrshl
addhn2	csel	fcvtns	frintz	ldursw	saba	sminv	sshr	tbnz	uqrshrn
addp	csinc	fcvtnu	frsqrte	ldxp	sabal	smlal	ssra	tbx	uqrshrn2
adds	csinv	fcvtps	frsqrts	ldxr	sabal2	smlal2	ssubl	tbz	uqshl
addv	csneg	fcvtpu	fsqrt	ldxrb	sabd	smlsl	ssubl2	trn1	uqshrn
adr	dcps1	fcvtxn	fsub	ldxrh	sabdl	smlsl2	ssubw	trn2	uqsub
adrp	dcps2	fcvtxn2	hint	lslv	sabdl2	smov	ssubw2	uaba	uqxtn
aesd	dcps3	fcvtzs	hlt	lsrv	sadalp	smsubl	st1	uabal	uqxtn2
aese	dmb	fcvtzu	hvc	madd	saddl	smulh	st2	uabal2	urecpe
aesimc	drps	fdiv	ins	mla	saddl2	smull	st3	uabd	urhadd
aesmc	dsb	fmadd	isb	mls	saddlp	smull2	st4	uabdl	urshl
and	dup	fmax	ld1	movi	saddlv	sqabs	stlr	uabdl2	urshr
ands	eon	fmaxnm	ld2	movk	saddw	sqadd	stlrb	uadalp	ursqrte
asrv	eor	fmaxnmp	ld3	movn	saddw2	sqdmlal	stlrh	uaddl	ursra
b	eret	fmaxnmv	ld4	movz	sbc	sqdmlal2	stlxp	uaddl2	ushl
b.c	ext	fmaxp	ldar	mrs	sbcs	sqdmlsl	stlxr	uaddlp	ushr
bfm	extr	fmaxv	ldarb	msr	sbfm	sqdmlsl2	stlxrb	uaddlv	usqadd
bic	fabd	fmin	ldarh	msub	scvtf	sqdmulh	stlxrh	uaddw	usra
bics	fabs	fminnm	ldaxp	mul	sdiv	sqdmull	stnp	uaddw2	usubl
bif	facge	fminnmp	ldaxr	mvni	sha1c	sqdmull2	stp	ubfm	usubl2
bit	facgt	fminnmv	ldaxrb	neg	sha1h	sqneg	str	ucvtf	usubw
bl	fadd	fminp	ldaxrh	not	sha1m	sqrdmulh	strb	udiv	usubw2
blr	faddp	fminv	ldnp	orn	sha1p	sqrshl	strh	uhadd	uzp1
br	fccmp	fmla	ldp	orr	sha1su0	sqrshrn	sttr	uhsub	uzp2
brk	fccmpe	fmls	ldpsw	pmul	sha1su1	sqrshrn2	sttrb	umaddl	xtn
bsl	fcmeq	fmov	ldr	pmull	sha256h	sqrshrun	sttrh	umax	xtn2
cbnz	fcmge	fmsub	ldrb	pmull2	sha256h2	sqrshrun2	stur	umaxp	zip1
cbz	fcmgt	fmul	ldrh	prfm	sha256su0	sqshl	sturb	umaxv	zip2
ccmn	fcmle	fmulx	ldrsb	prfum	sha256su1	sqshlu	sturh	umin	
ccmp	fcmlt	fneg	ldrsh	raddhn	shadd	sqshrn	stxp	uminp	
clrex	fcmp	fnmadd	ldrsw	raddhn2	shl	sqshrun	stxr	uminv	
cls	fcmpe	fnmsub	ldtr	rbit	shll	sqsub	stxrb	umlal	
clz	fcsel	fnmul	ldtrb	ret	shll2	sqxtn	stxrh	umlal2	
cmeq	fcvt	frecpe	ldtrh	rev	shsub	sqxtn2	sub	umlsl	
cmge	fcvtas	frecps	ldtrsb	rev16	sli	sqxtun	subhn	umlsl2	
cmgt	fcvtau	frecpx	ldtrsh	rev32	smaddl	sqxtun2	subhn2	umov	
cmhi	fcvtl	frinta	ldtrsw	rev64	smax	srhadd	subs	umsubl	

http://kitoslab-eng.blogspot.com/2012/10/armv8-aarch64-instruction-encoding.html

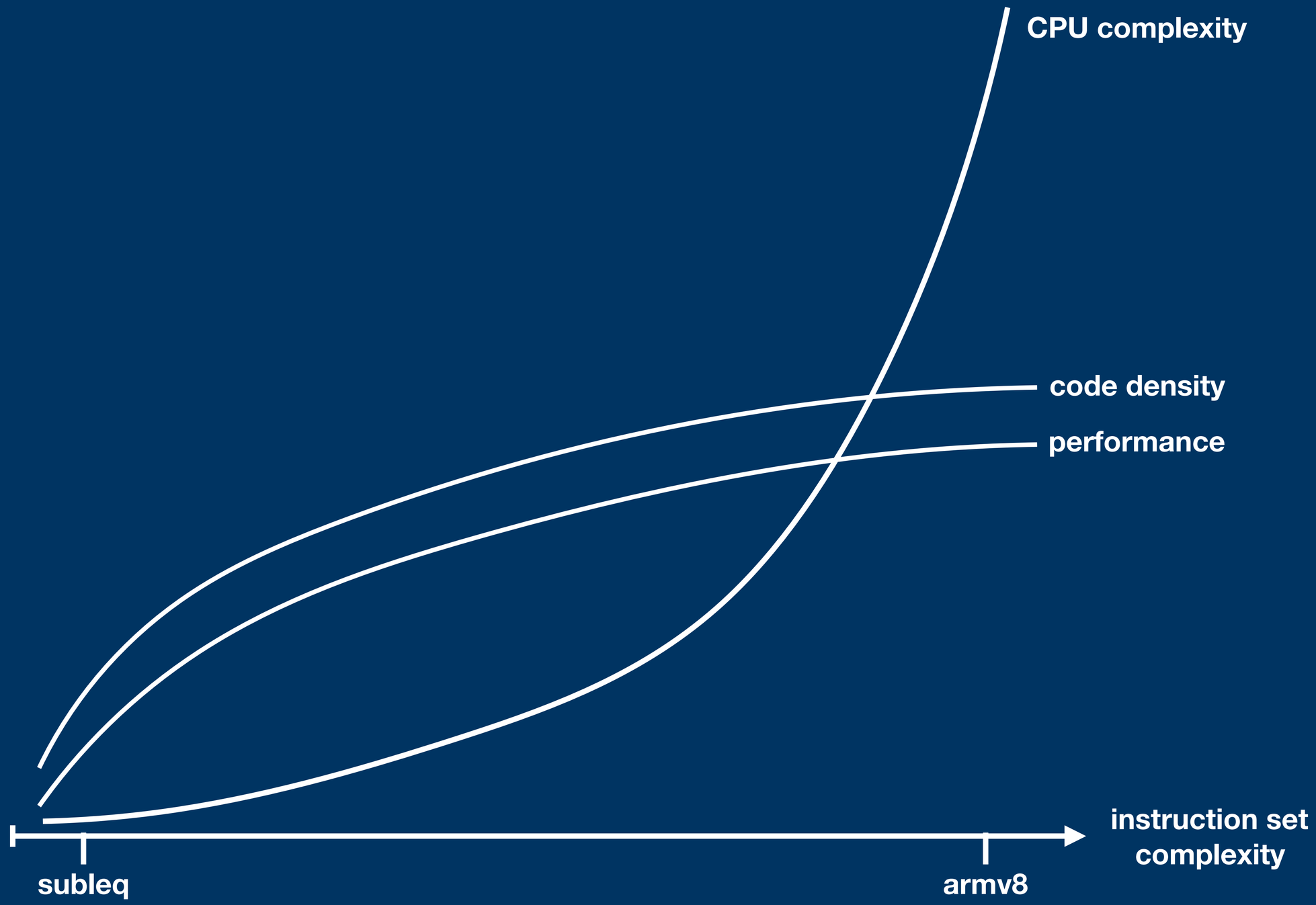
subleq

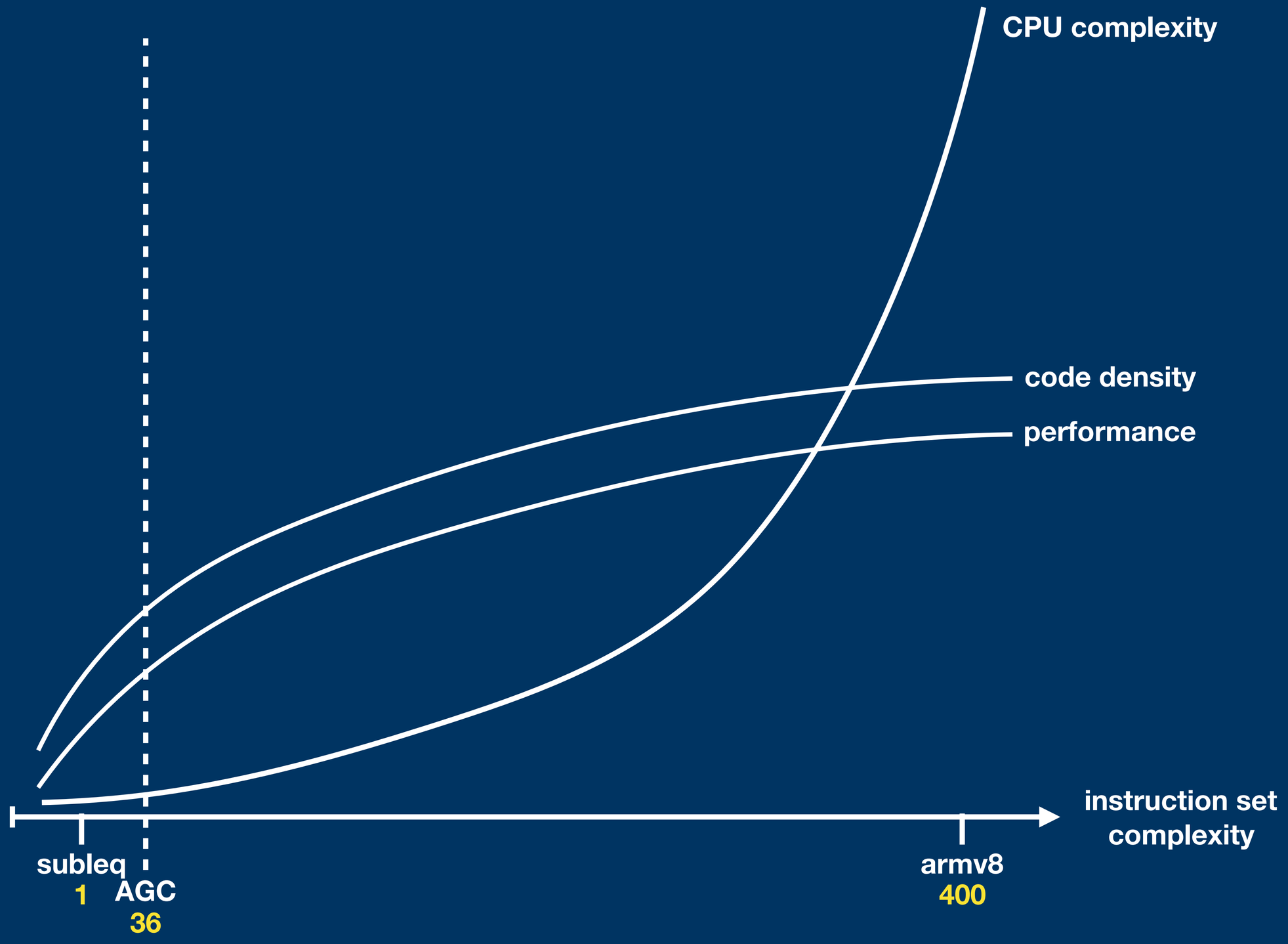
armv8


subleq
armv8
instruction set
complexity


code density
subleq
armv8
instruction set
complexity


code density
performance
instruction set
complexity
subleq
armv8


CPU complexity
code density
performance
instruction set
complexity
subleq
armv8


CPU complexity
code density
performance
instruction set
complexity
subleq
1
AGC
36
armv8
400

Load/Store

Load Cpl	ldc a, [k]	CS k
	4000+k	2
	A ⟵ -M[k]	

Load	ld a, [k]	CA k
	3000+k	2
	A ⟵ M[k]	

Exchange	xchg a, [k]	XCH k
	5C00+k	2
	A ⟷ M[k]	

Store	ld [k], a	TS k
	5800+k	2
	M[k] ⟵ A	

Load Dbl Cpl	ldc ab, [k]	DCS k
	0006, 4001+k	4
	A ⟵ -M[k] B ⟵ -M[k+1]	

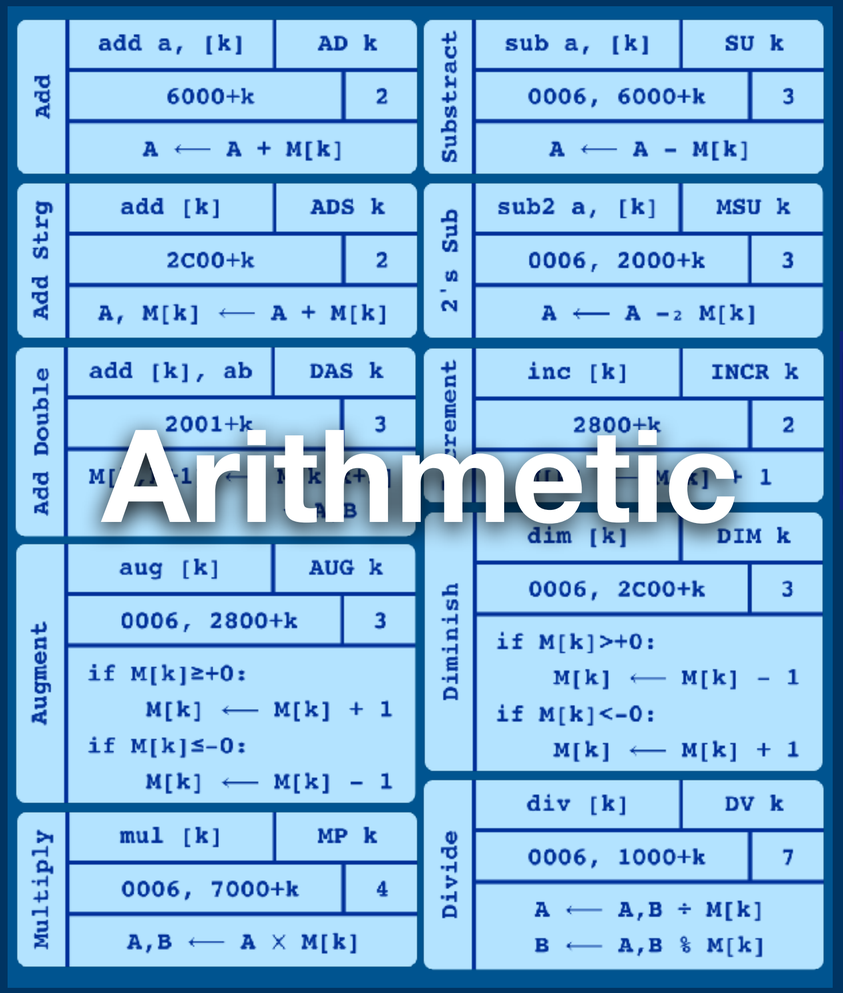
Load Doub	0006, 3001+k	4
	A ⟵ M[k] B ⟵ M[k+1]	

Xchg Dbl	0006, 5201+k	3
	A ⟷ M[k] B ⟷ M[k+1]	

Xchg B	xchg b, [k]	LXCH k
	2400+k	2
	B ⟷ M[k]	

Xchg LR	xchg lr, [k]	QXCH k
	0600, 2400+k	3
	LR ⟷ M[k]	


Load Cpl
ldc a, [k]
CS k
4000+k
2
A ⟵ -M[k]
Load Dbl Cpl
ldc ab, [k]
DCS k
0006, 4001+k
4
A ⟵ -M[k]
B ⟵ -M[k+1]
Load
ld a, [k]
CA k
3000+k
2
A ⟵ M[k]
Load Doub
0006, 3001+k
4
A ⟵ M[k]
B ⟵ M[k+1]
Load/Store
Xchange
xchg a, [k]
XCH k
5C00+k
2
A ⟷ M[k]
Xchg Dbl
DXCH k
5201+k
3
A ⟷ M[k]
B ⟷ M[k+1]
Xchg B
xchg b, [k]
LXCH k
2400+k
2
B ⟷ M[k]
Xchg LR
xchg lr, [k]
QXCH k
0600, 2400+k
3
LR ⟷ M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ⟵ A


Add
add a, [k]
AD k
6000+k
2
A ⟵ A + M[k]
Add Strg
add [k]
ADS k
2C00+k
2
A, M[k] ⟵ A + M[k]
Add Double
add [k], ab
DAS k
2001+k
3
Augment
aug [k]
AUG k
0006, 2800+k
3
if M[k]≥+0:
M[k] ⟵ M[k] + 1
if M[k]≤-0:
M[k] ⟵ M[k] - 1
Multiply
mul [k]
MP k
0006, 7000+k
4
A,B ⟵ A × M[k]
Substract
sub a, [k]
SU k
0006, 6000+k
3
A ⟵ A - M[k]
2's Sub
sub2 a, [k]
MSU k
0006, 2000+k
3
A ⟵ A -₂ M[k]
Diminish
dim [k]
DIM k
0006, 2C00+k
3
if M[k]>+0:
M[k] ⟵ M[k] - 1
if M[k]<-0:
M[k] ⟵ M[k] + 1
Divide
div [k]
DV k
0006, 1000+k
7
A ⟵ A,B ÷ M[k]
B ⟵ A,B % M[k]
Arithmetic


AND
and a, [k]
MASK k
A ⟵ M[k]
Logic

Load/Store

Load Cpl	ldc a, [k]	CS k
	4000+k	2
	A ⟵ -M[k]	

Load Dbl Cpl	ldc ab, [k]	DCS k
	0006, 4001+k	4
	A ⟵ -M[k] B ⟵ -M[k+1]	

Load	ld a, [k]	CA k
	3000+k	2
	A ⟵ M[k]	

Load Doub	[illegible]	[illegible] k
	0006, 3001+k	4
	A ⟵ M[k] B ⟵ M[k+1]	

Xchange	xchg a, [k]	XCH k
	5C00+k	2
	A ⟷ M[k]	

Xchg Dbl	[illegible]	DXCH k
	5201+k	3
	A ⟷ M[k] B ⟷ M[k+1]	

Xchg B	xchg b, [k]	LXCH k
	2400+k	2
	B ⟷ M[k]	

Xchg LR	xchg lr, [k]	QXCH k
	0600, 2400+k	3
	LR ⟷ M[k]	

Store	ld [k], a	TS k
	5800+k	2
	M[k] ⟵ A	

Arithmetic

Add	add a, [k]	AD k
	6000+k	2
	A ⟵ A + M[k]	

Add Strg	add [k]	ADS k
	2C00+k	2
	A, M[k] ⟵ A + M[k]	

Add Double	add [k], ab	DAS k
	2001+k	3
	M[[illegible]	

Substract	sub a, [k]	SU k
	0006, 6000+k	3
	A ⟵ A - M[k]	

2's Sub	sub2 a, [k]	MSU k
	0006, 2000+k	3
	A ⟵ A -₂ M[k]	

[illegible]rement	inc [k]	INCR k
	2800+k	2
	[illegible]	

Augment	aug [k]	AUG k
	0006, 2800+k	3

	if M[k]≥+0:
	M[k] ⟵ M[k] + 1
	if M[k]≤-0:
	M[k] ⟵ M[k] - 1	

Diminish	dim [k]	DIM k
	0006, 2C00+k	3

	if M[k]>+0:
	M[k] ⟵ M[k] - 1
	if M[k]<-0:
	M[k] ⟵ M[k] + 1	

Multiply	mul [k]	MP k
	0006, 7000+k	4
	A,B ⟵ A × M[k]	

Divide	div [k]	DV k
	0006, 1000+k	7
	A ⟵ A,B ÷ M[k] B ⟵ A,B % M[k]	

Logic

AND	and a, [k]	MASK k
	[illegible]	
	A ⟵ [illegible] M[k]	

Control Flow

Jump	jmp k	TCF k
	1000+k	1
	PC ⟵ k	

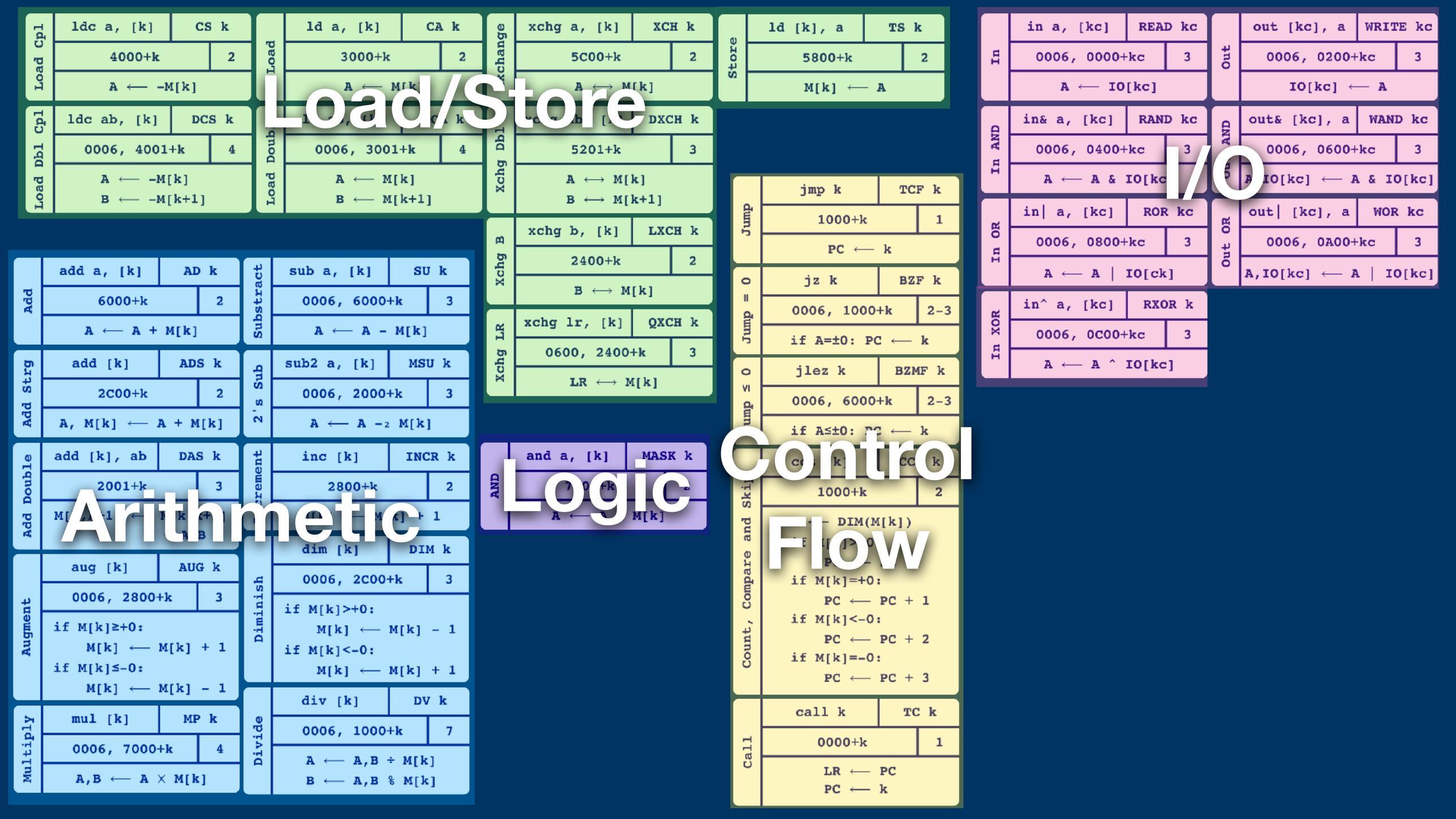
Jump = 0	jz k	BZF k
	0006, 1000+k	2-3
	if A=±0: PC ⟵ k	

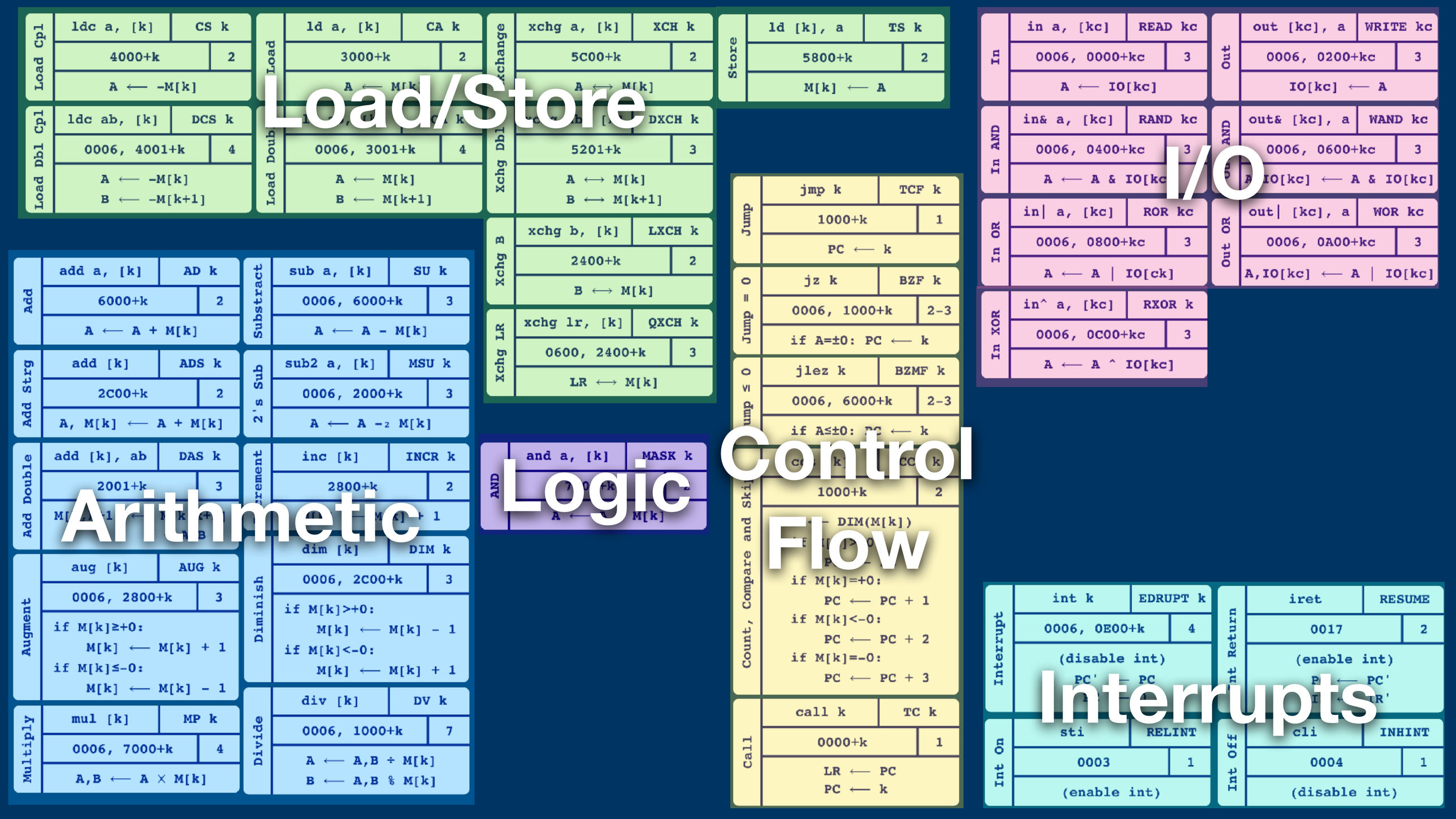
Jump ≤ 0	jlez k	BZMF k
	0006, 6000+k	2-3
	if A≤±0: [illegible]	

Count, Compare and Ski[illegible]	[illegible] k	[illegible] k
	1000+k	2

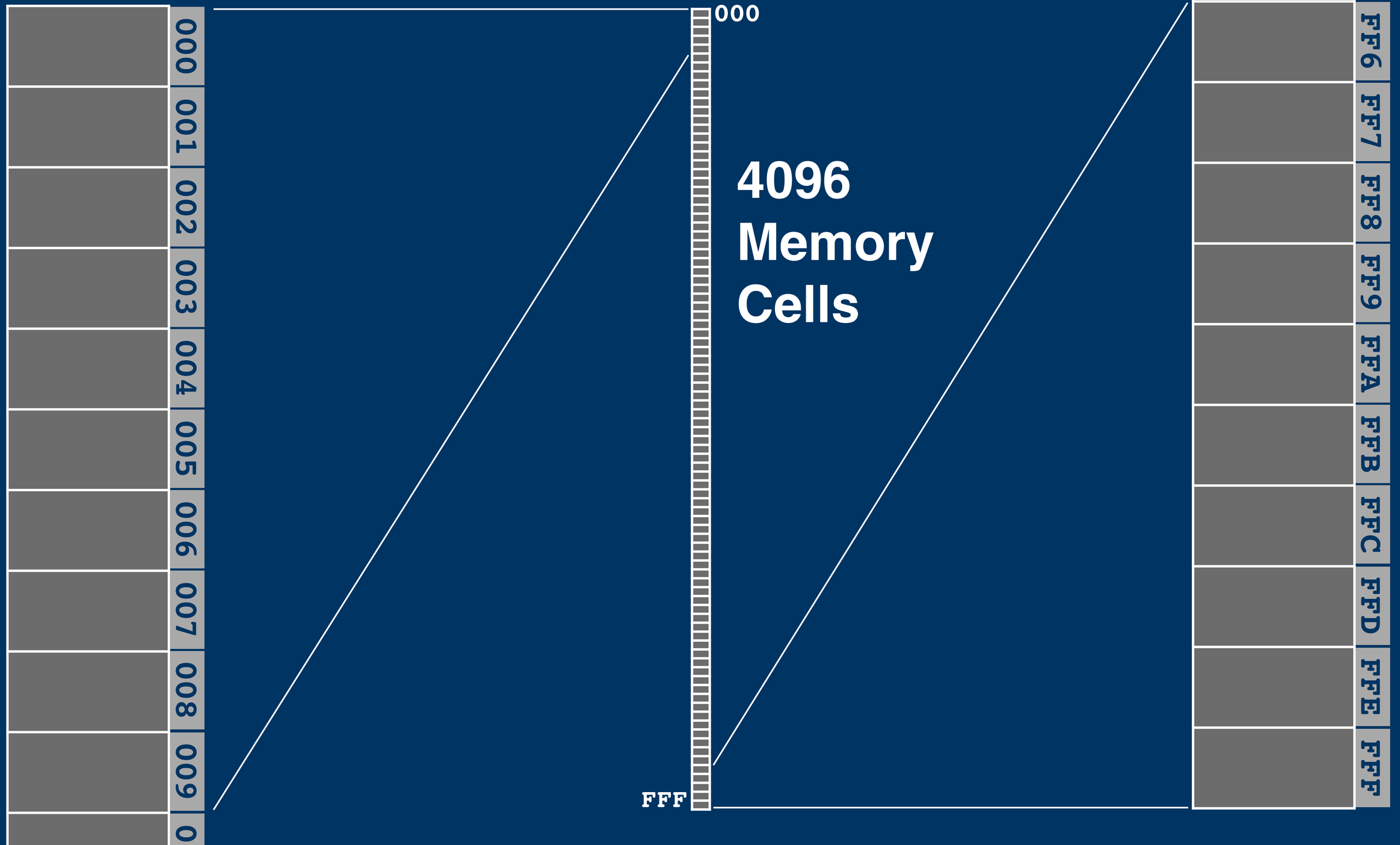
	[illegible] DIM(M[k])
	[illegible]
	if M[k]=+0:
	PC ⟵ PC + 1
	if M[k]<-0:
	PC ⟵ PC + 2
	if M[k]=-0:
	PC ⟵ PC + 3	

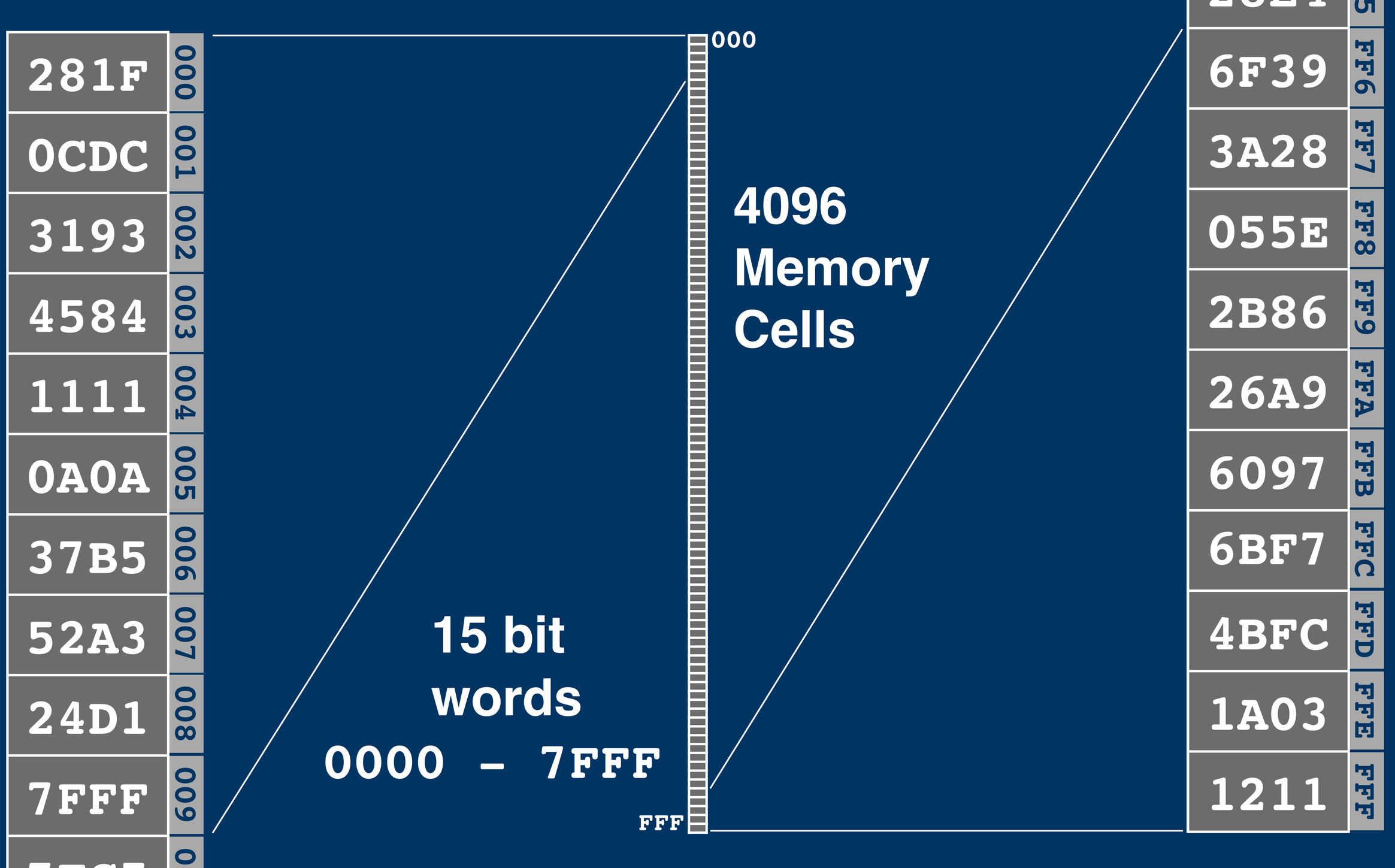
Call	call k	TC k
	0000+k	1
	LR ⟵ PC PC ⟵ k	

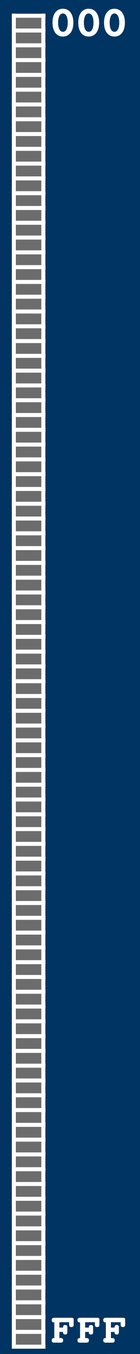

Load/Store
Load Cpl
ldc a, [k]
CS k
4000+k
2
A ⟵ -M[k]
Load Dbl Cpl
ldc ab, [k]
DCS k
0006, 4001+k
4
A ⟵ -M[k]
B ⟵ -M[k+1]
Load
ld a, [k]
CA k
3000+k
2
A ⟵ M[k]
Load Doub
0006, 3001+k
4
A ⟵ M[k]
B ⟵ M[k+1]
Xchange
xchg a, [k]
XCH k
5C00+k
2
A ⟷ M[k]
Xchg Dbl
DXCH k
5201+k
3
A ⟷ M[k]
B ⟷ M[k+1]
Xchg B
xchg b, [k]
LXCH k
2400+k
2
B ⟷ M[k]
Xchg LR
xchg lr, [k]
QXCH k
0600, 2400+k
3
LR ⟷ M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ⟵ A
I/O
In
in a, [kc]
READ kc
0006, 0000+kc
3
A ⟵ IO[kc]
Out
out [kc], a
WRITE kc
0006, 0200+kc
3
IO[kc] ⟵ A
In AND
in& a, [kc]
RAND kc
0006, 0400+kc
3
A ⟵ A & IO[kc]
Out AND
out& [kc], a
WAND kc
0006, 0600+kc
3
IO[kc] ⟵ A & IO[kc]
In OR
in| a, [kc]
ROR kc
0006, 0800+kc
3
A ⟵ A | IO[ck]
Out OR
out| [kc], a
WOR kc
0006, 0A00+kc
3
A,IO[kc] ⟵ A | IO[kc]
In XOR
in^ a, [kc]
RXOR k
0006, 0C00+kc
3
A ⟵ A ^ IO[kc]
Arithmetic
Add
add a, [k]
AD k
6000+k
2
A ⟵ A + M[k]
Substract
sub a, [k]
SU k
0006, 6000+k
3
A ⟵ A - M[k]
Add Strg
add [k]
ADS k
2C00+k
2
A, M[k] ⟵ A + M[k]
2's Sub
sub2 a, [k]
MSU k
0006, 2000+k
3
A ⟵ A -2 M[k]
Add Double
add [k], ab
DAS k
2001+k
3
Augment
aug [k]
AUG k
0006, 2800+k
3
if M[k]≥+0:
M[k] ⟵ M[k] + 1
if M[k]≤-0:
M[k] ⟵ M[k] - 1
Diminish
dim [k]
DIM k
0006, 2C00+k
3
if M[k]>+0:
M[k] ⟵ M[k] - 1
if M[k]<-0:
M[k] ⟵ M[k] + 1
Multiply
mul [k]
MP k
0006, 7000+k
4
A,B ⟵ A × M[k]
Divide
div [k]
DV k
0006, 1000+k
7
A ⟵ A,B ÷ M[k]
B ⟵ A,B % M[k]
Logic
AND
and a, [k]
MASK k
A ⟵ M[k]
Control Flow
Jump
jmp k
TCF k
1000+k
1
PC ⟵ k
Jump = 0
jz k
BZF k
0006, 1000+k
2-3
if A=±0: PC ⟵ k
Count, Compare and Skip
1000+k
2
if M[k]=+0:
PC ⟵ PC + 1
if M[k]<-0:
PC ⟵ PC + 2
if M[k]=-0:
PC ⟵ PC + 3
Call
call k
TC k
0000+k
1
LR ⟵ PC
PC ⟵ k


Load/Store
Load Cpl
ldc a, [k]
CS k
4000+k
2
A ⟵ -M[k]
Load Dbl Cpl
ldc ab, [k]
DCS k
0006, 4001+k
4
A ⟵ -M[k]
B ⟵ -M[k+1]
Load
ld a, [k]
CA k
3000+k
2
A ⟵ M[k]
Load Doub
0006, 3001+k
4
A ⟵ M[k]
B ⟵ M[k+1]
Exchange
xchg a, [k]
XCH k
5C00+k
2
A ⟷ M[k]
Xchg Dbl
DXCH k
5201+k
3
A ⟷ M[k]
B ⟷ M[k+1]
Xchg B
xchg b, [k]
LXCH k
2400+k
2
B ⟷ M[k]
Xchg LR
xchg lr, [k]
QXCH k
0600, 2400+k
3
LR ⟷ M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ⟵ A
I/O
In
in a, [kc]
READ kc
0006, 0000+kc
3
A ⟵ IO[kc]
Out
out [kc], a
WRITE kc
0006, 0200+kc
3
IO[kc] ⟵ A
In AND
in& a, [kc]
RAND kc
0006, 0400+kc
3
A ⟵ A & IO[kc]
Out AND
out& [kc], a
WAND kc
0006, 0600+kc
3
IO[kc] ⟵ A & IO[kc]
In OR
in| a, [kc]
ROR kc
0006, 0800+kc
3
A ⟵ A | IO[ck]
Out OR
out| [kc], a
WOR kc
0006, 0A00+kc
3
A,IO[kc] ⟵ A | IO[kc]
In XOR
in^ a, [kc]
RXOR k
0006, 0C00+kc
3
A ⟵ A ^ IO[kc]
Arithmetic
Add
add a, [k]
AD k
6000+k
2
A ⟵ A + M[k]
Substract
sub a, [k]
SU k
0006, 6000+k
3
A ⟵ A - M[k]
Add Strg
add [k]
ADS k
2C00+k
2
A, M[k] ⟵ A + M[k]
2's Sub
sub2 a, [k]
MSU k
0006, 2000+k
3
A ⟵ A -₂ M[k]
Add Double
add [k], ab
DAS k
2001+k
3
Increment
inc [k]
INCR k
2800+k
2
Augment
aug [k]
AUG k
0006, 2800+k
3
if M[k]≥+0:
M[k] ⟵ M[k] + 1
if M[k]≤-0:
M[k] ⟵ M[k] - 1
Diminish
dim [k]
DIM k
0006, 2C00+k
3
if M[k]>+0:
M[k] ⟵ M[k] - 1
if M[k]<-0:
M[k] ⟵ M[k] + 1
Multiply
mul [k]
MP k
0006, 7000+k
4
A,B ⟵ A × M[k]
Divide
div [k]
DV k
0006, 1000+k
7
A ⟵ A,B ÷ M[k]
B ⟵ A,B % M[k]
Logic
AND
and a, [k]
MASK k
A ⟵ A & M[k]
Control Flow
Jump
jmp k
TCF k
1000+k
1
PC ⟵ k
Jump = 0
jz k
BZF k
0006, 1000+k
2-3
if A=±0: PC ⟵ k
Jump ≤ 0
jlez k
BZMF k
0006, 6000+k
2-3
if A≤±0: PC ⟵ k
Count, Compare and Skip
1000+k
2
⟵ DIM(M[k])
if M[k]=+0:
PC ⟵ PC + 1
if M[k]<-0:
PC ⟵ PC + 2
if M[k]=-0:
PC ⟵ PC + 3
Call
call k
TC k
0000+k
1
LR ⟵ PC
PC ⟵ k
Interrupts
Interrupt
int k
EDRUPT k
0006, 0E00+k
4
(disable int)
Int Return
iret
RESUME
0017
2
(enable int)
Int On
sti
RELINT
0003
1
(enable int)
Int Off
cli
INHINT
0004
1
(disable int)

4096
Memory
Cells


000
001
002
003
004
005
006
007
008
009
FF5
FF6
FF7
FF8
FF9
FFA
FFB
FFC
FFD
FFE
FFF
000
FFF
4096
Memory
Cells

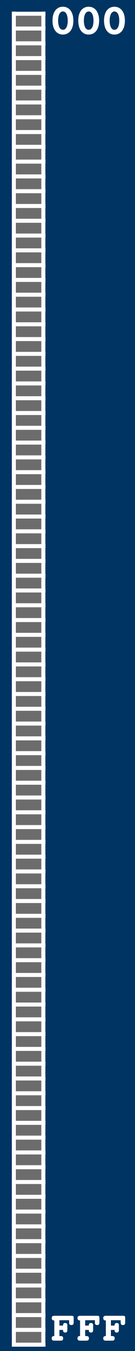

281F
0CDC
3193
4584
1111
0A0A
37B5
52A3
24D1
7FFF
000
001
002
003
004
005
006
007
008
009
000
FFF
4096
Memory
Cells
15 bit
words
0000 - 7FFF
6F39
3A28
055E
2B86
26A9
6097
6BF7
4BFC
1A03
1211
FF6
FF7
FF8
FF9
FFA
FFB
FFC
FFD
FFE
FFF

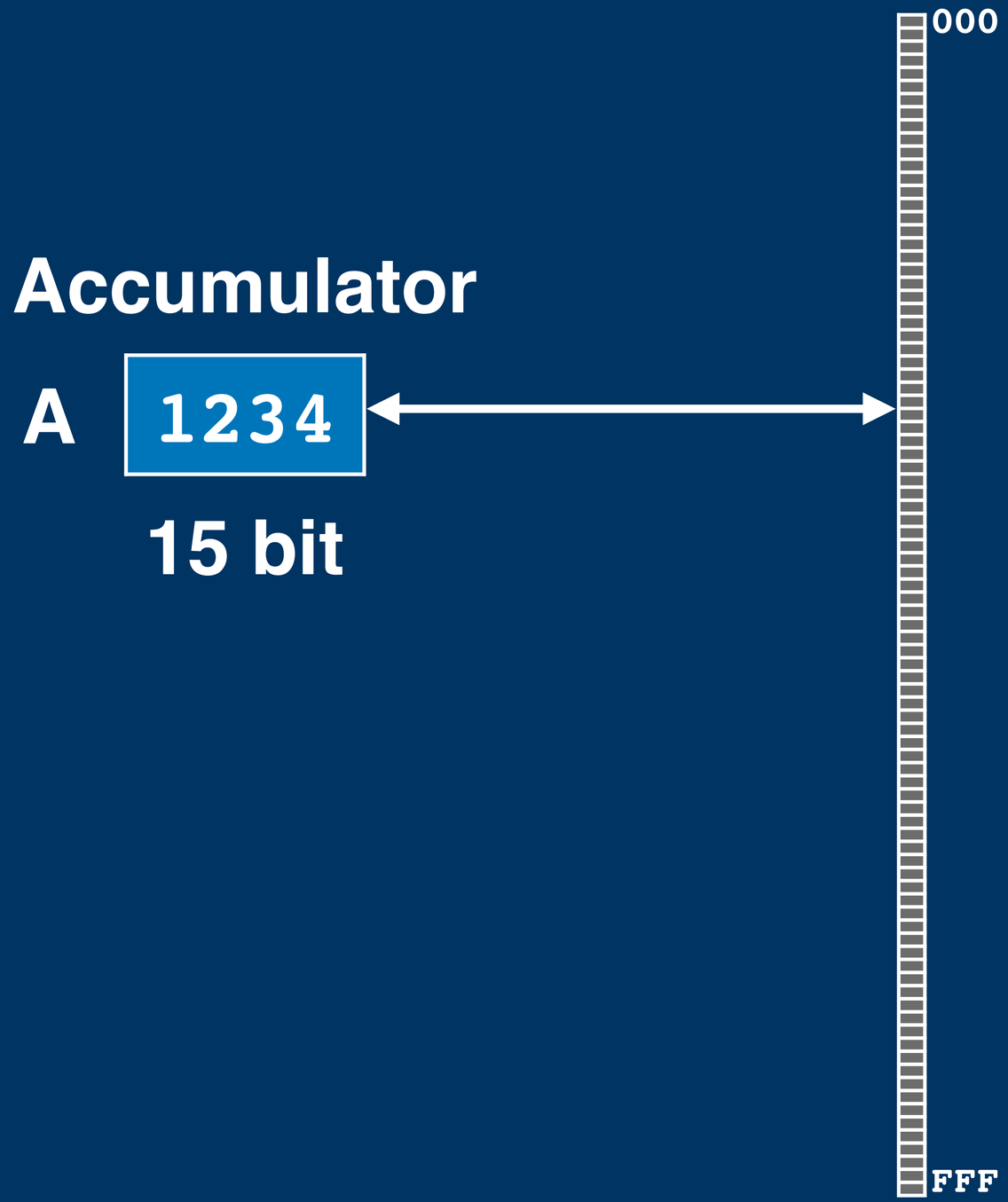

000
FFF

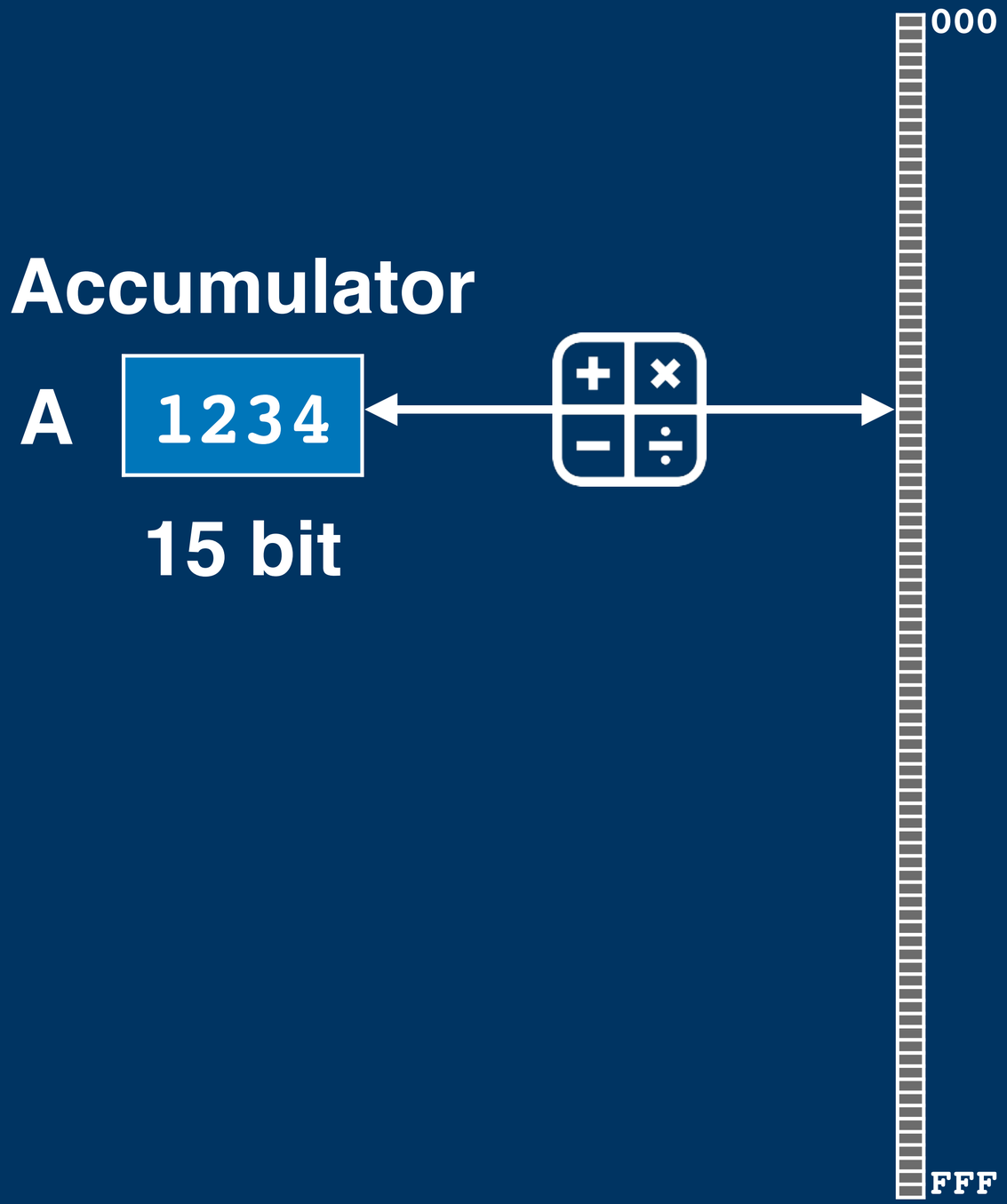
Accumulator

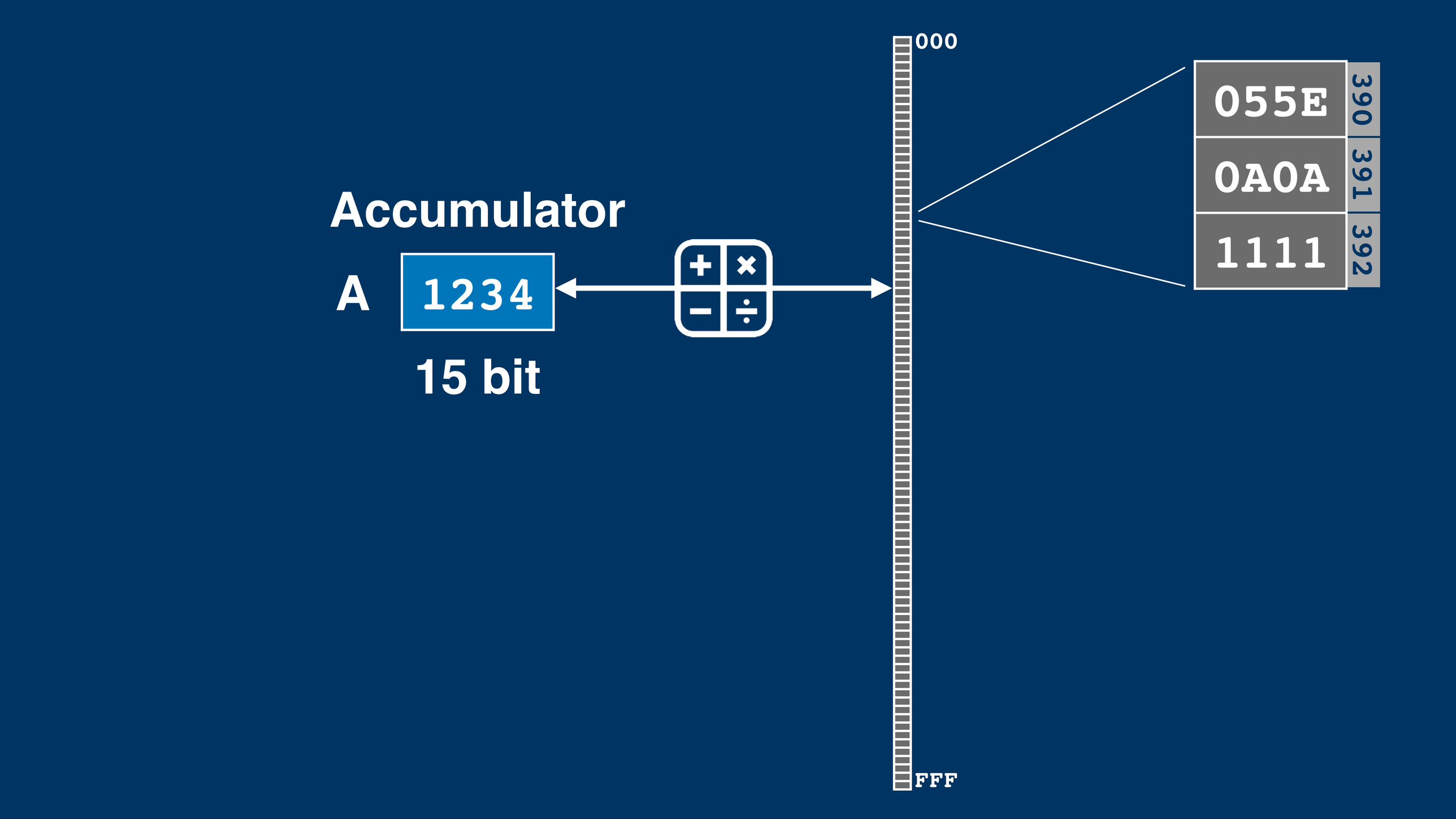
A `1234`

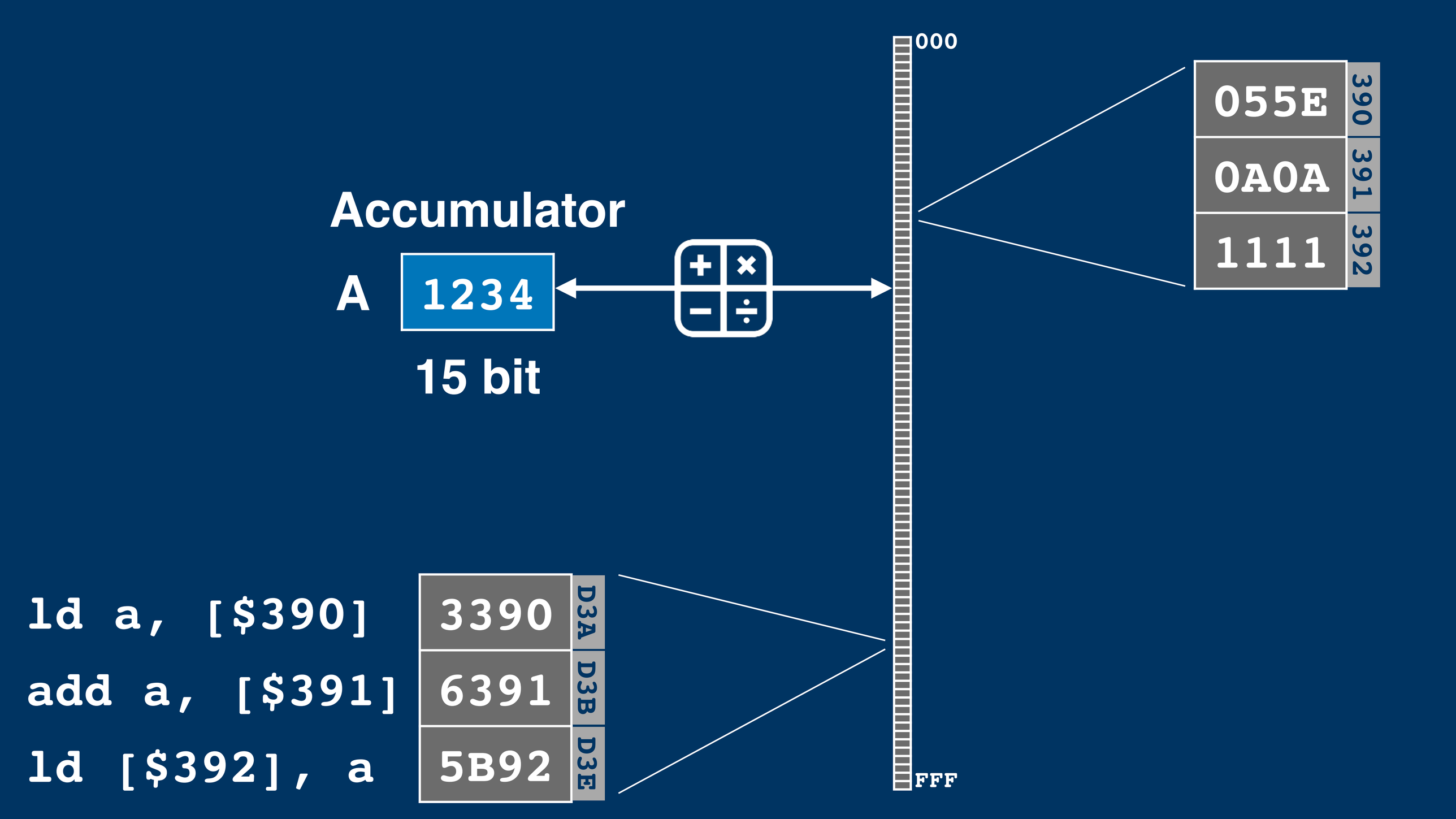
15 bit

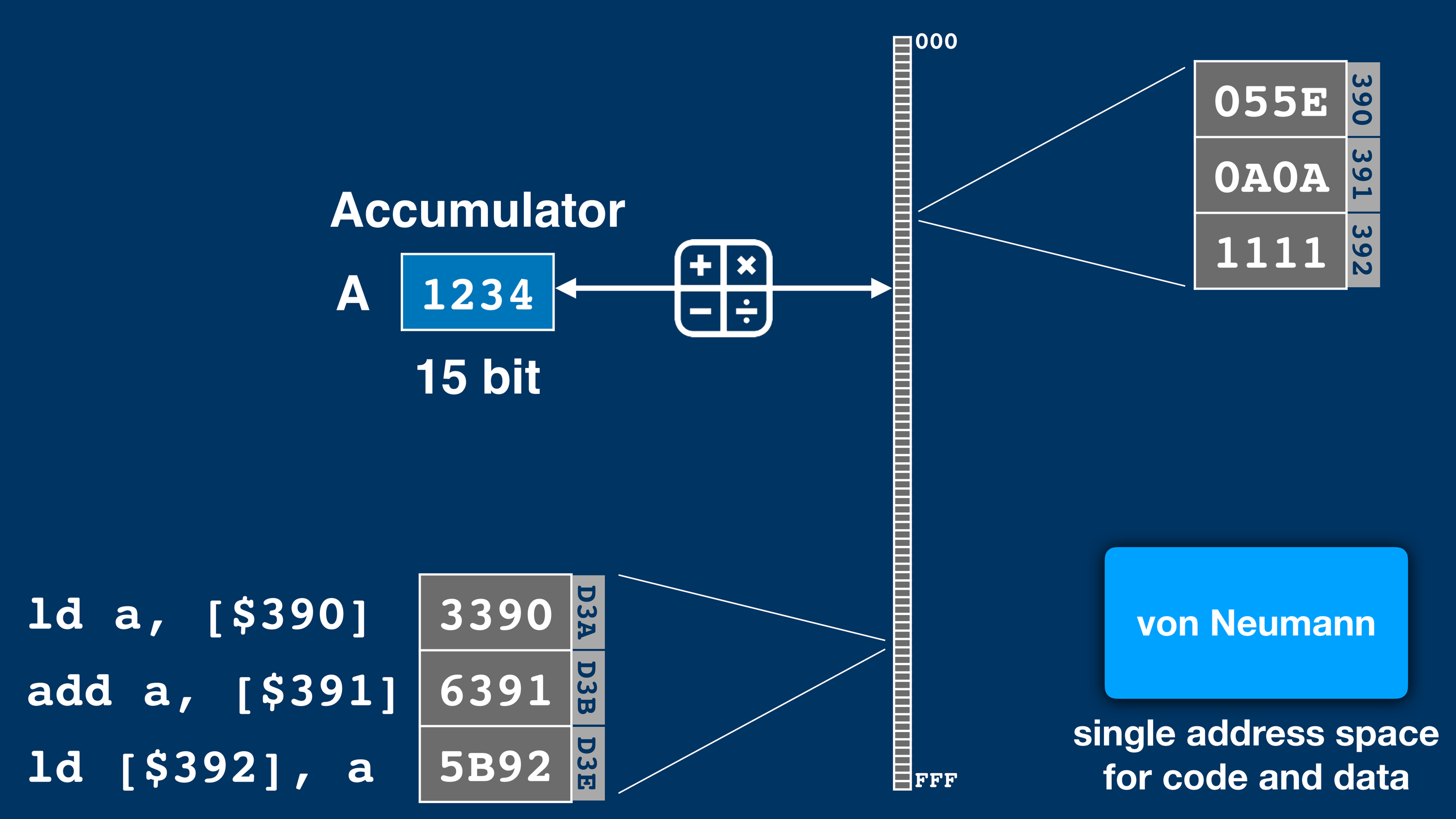


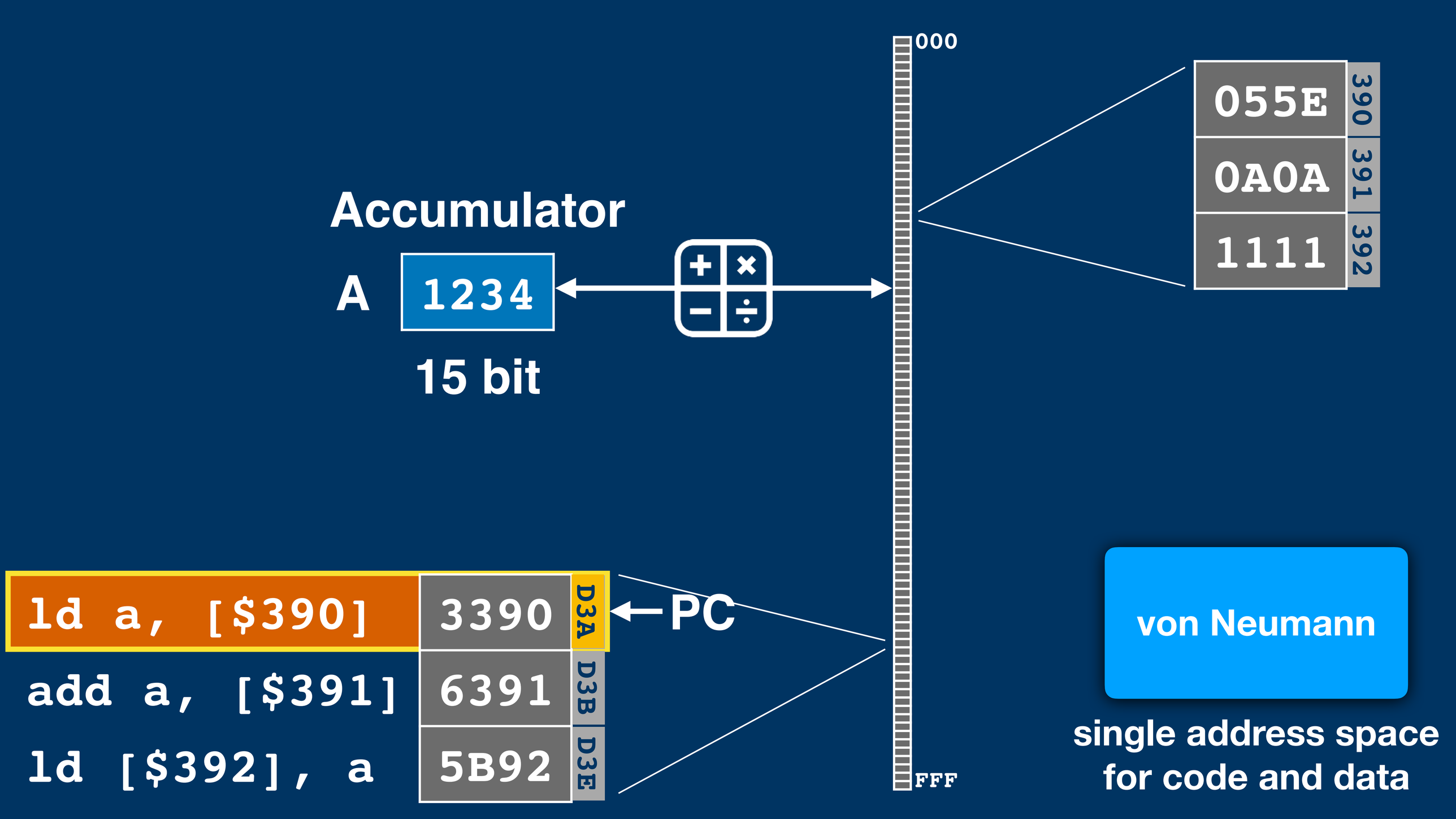

Accumulator
A
1234
15 bit
000
FFF

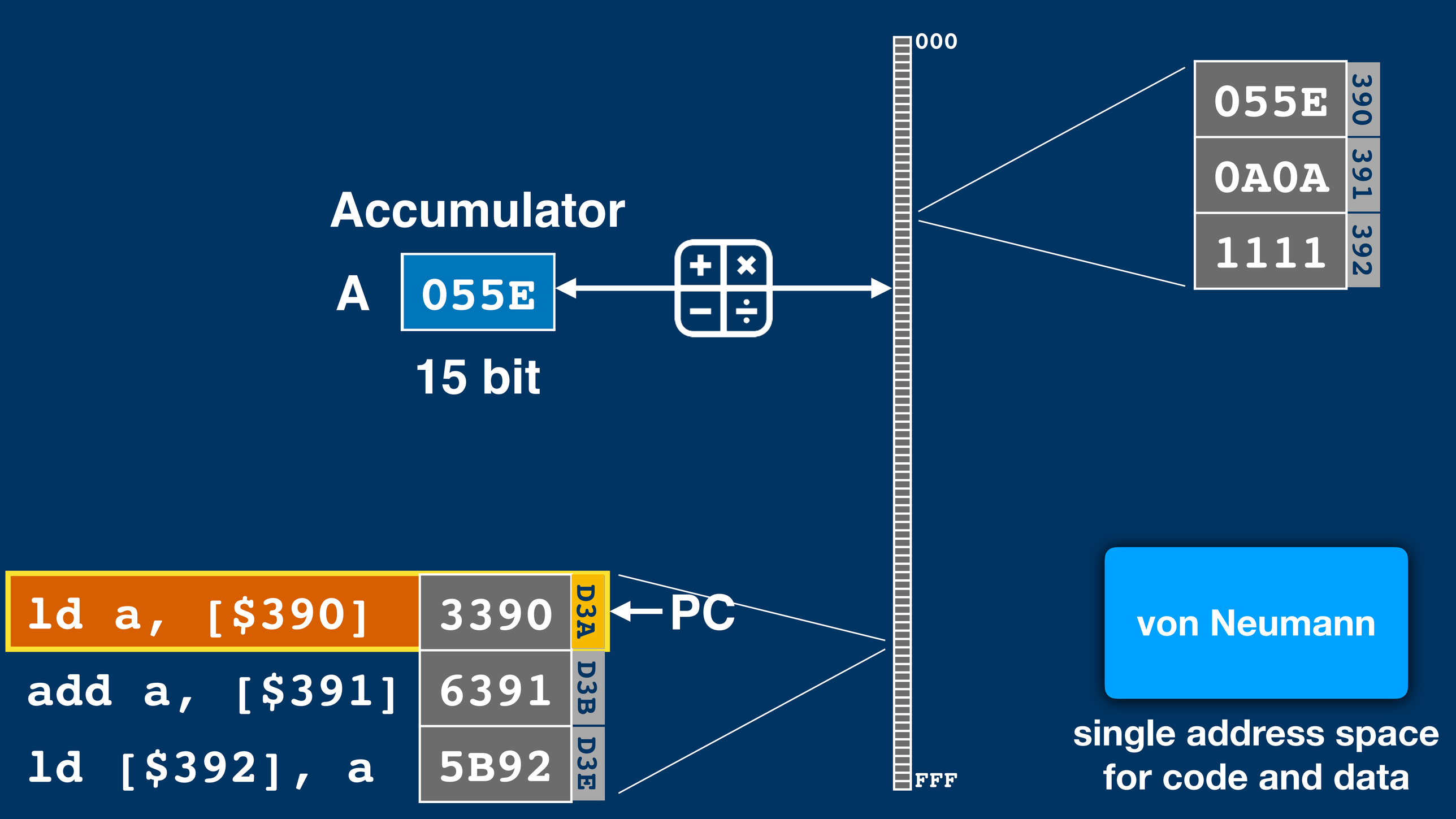

Accumulator
A
1234
15 bit
000
FFF

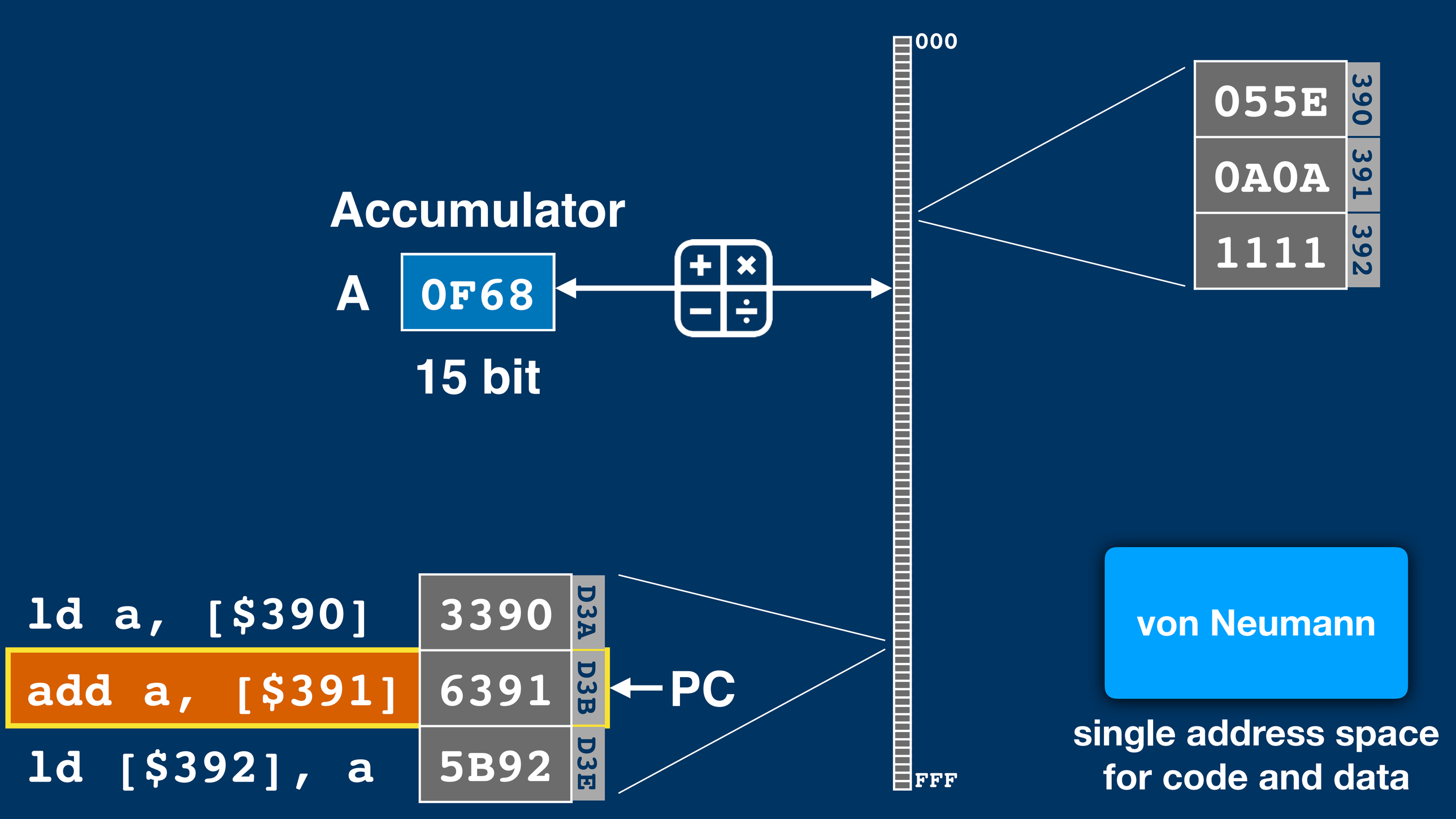

Accumulator
A
1234
15 bit
000
FFF
055E
0A0A
1111
390
391
392

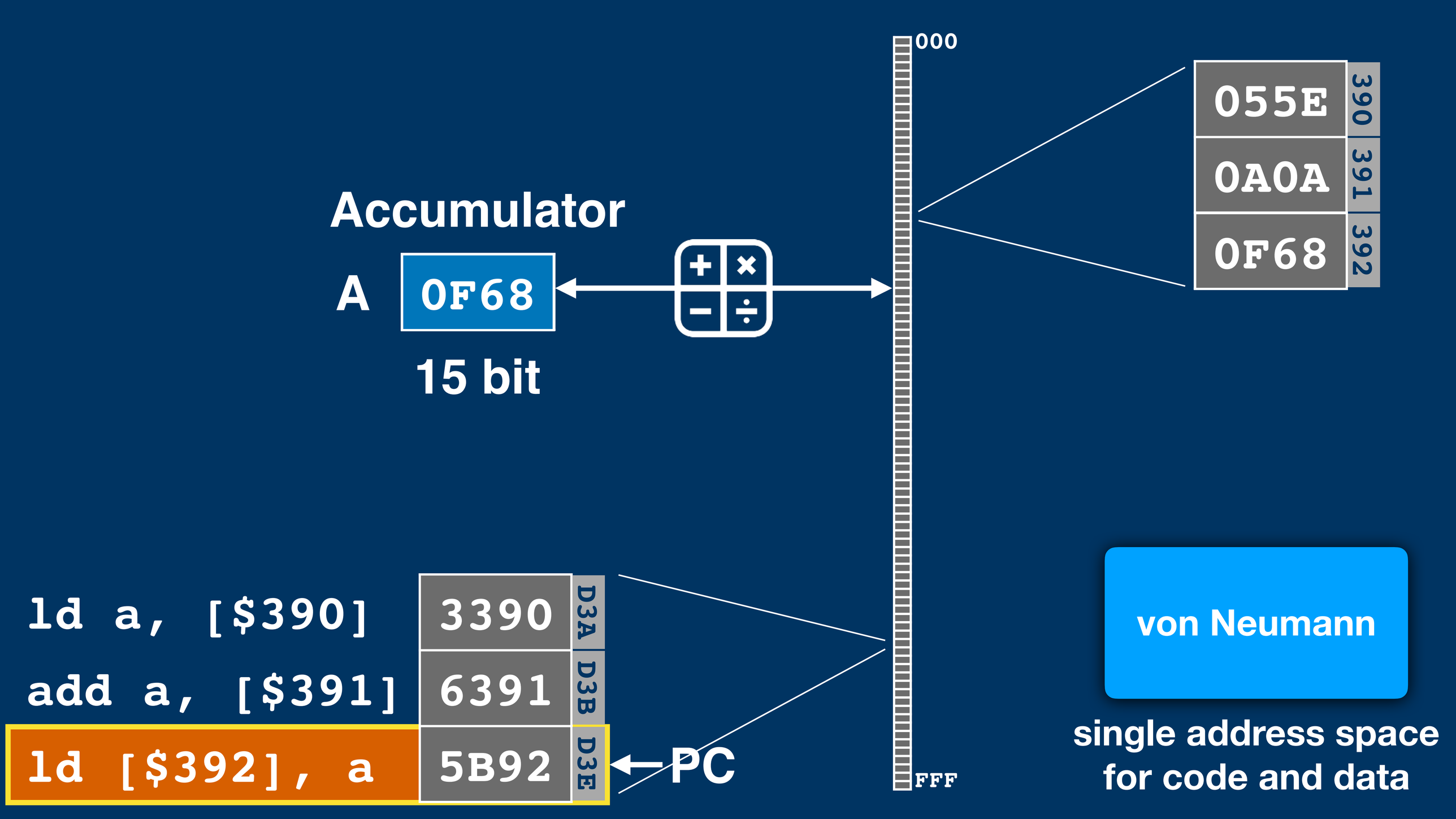

Accumulator
A
1234
15 bit
000
FFF
055E
0A0A
1111
390
391
392
ld a, [$390]
add a, [$391]
ld [$392], a
3390
6391
5B92
D3A
D3B
D3E


Accumulator
A
1234
15 bit
000
FFF
055E
0A0A
1111
390
391
392
ld a, [$390]
add a, [$391]
ld [$392], a
3390
6391
5B92
D3A
D3B
D3E
von Neumann
single address space
for code and data


Accumulator
A
1234
15 bit
000
FFF
055E
0A0A
1111
390
391
392
ld a, [$390]
3390
D3A
PC
add a, [$391]
6391
D3B
ld [$392], a
5B92
D3E
von Neumann
single address space
for code and data


Accumulator
A
055E
15 bit
000
FFF
055E
0A0A
1111
390
391
392
ld a, [$390]
3390
D3A
PC
add a, [$391]
6391
D3B
ld [$392], a
5B92
D3E
von Neumann
single address space
for code and data


Accumulator
A
0F68
15 bit
000
FFF
055E
0A0A
1111
390
391
392
ld a, [$390]
add a, [$391]
ld [$392], a
3390
6391
5B92
D3A
D3B
D3E
PC
von Neumann
single address space
for code and data


Accumulator
A
0F68
15 bit
000
FFF
055E
0A0A
0F68
390
391
392
ld a, [$390]
add a, [$391]
ld [$392], a
3390
6391
5B92
D3A
D3B
D3E
PC
von Neumann
single address space
for code and data

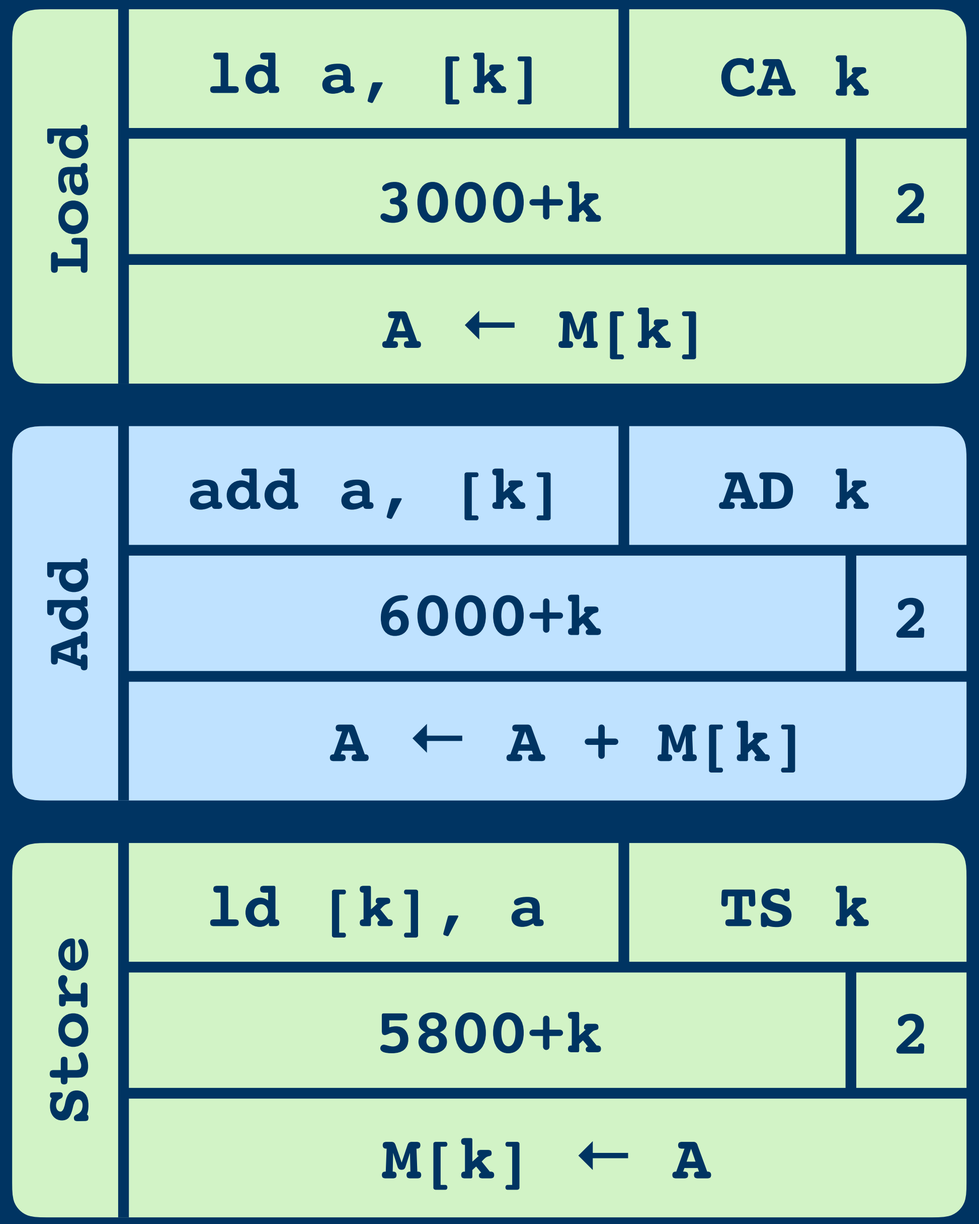
```
ld a, [$390]
add a, [$391]
ld [$392], a
```

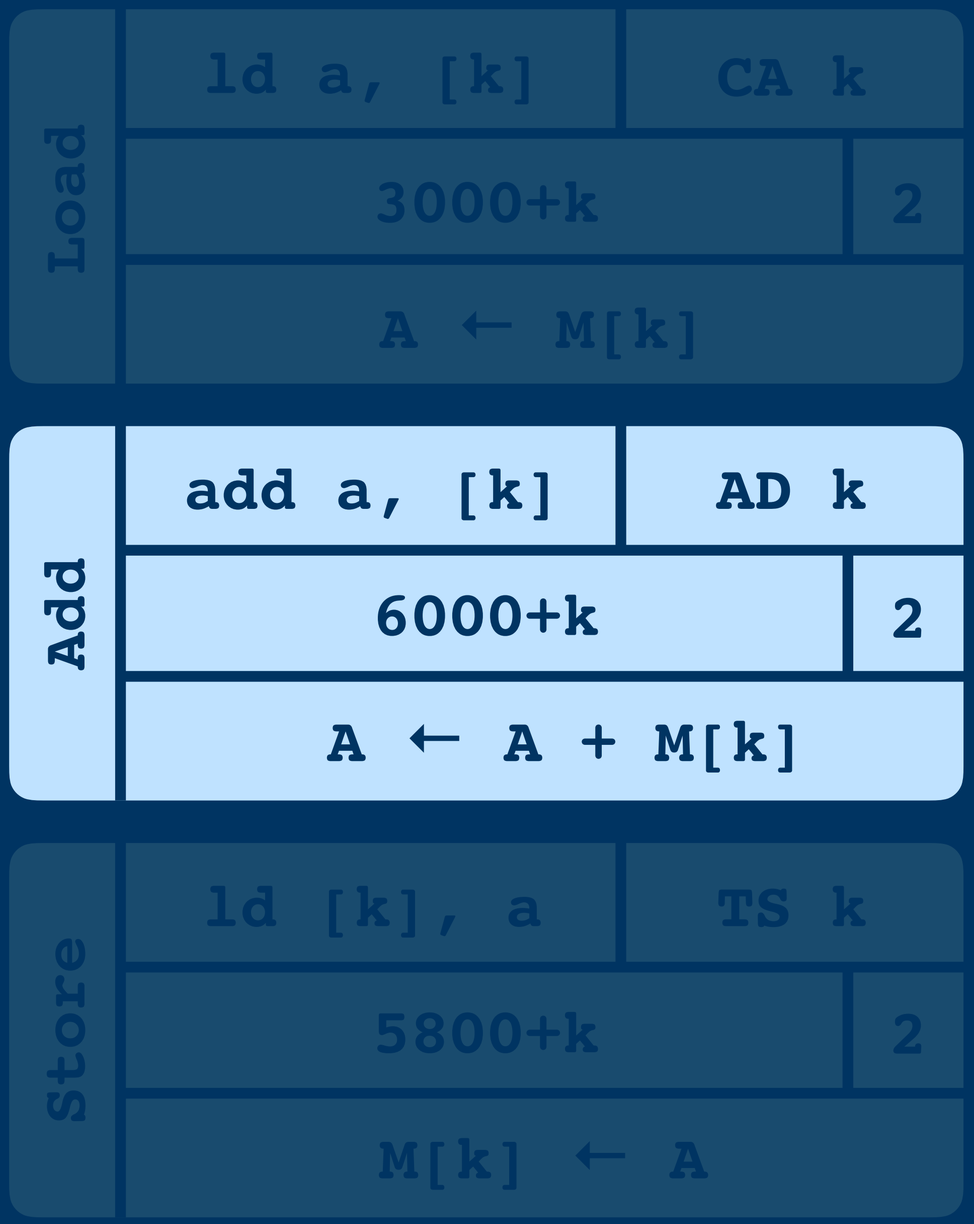
```
ld a, [k]
add a, [k]
ld [k], a
```

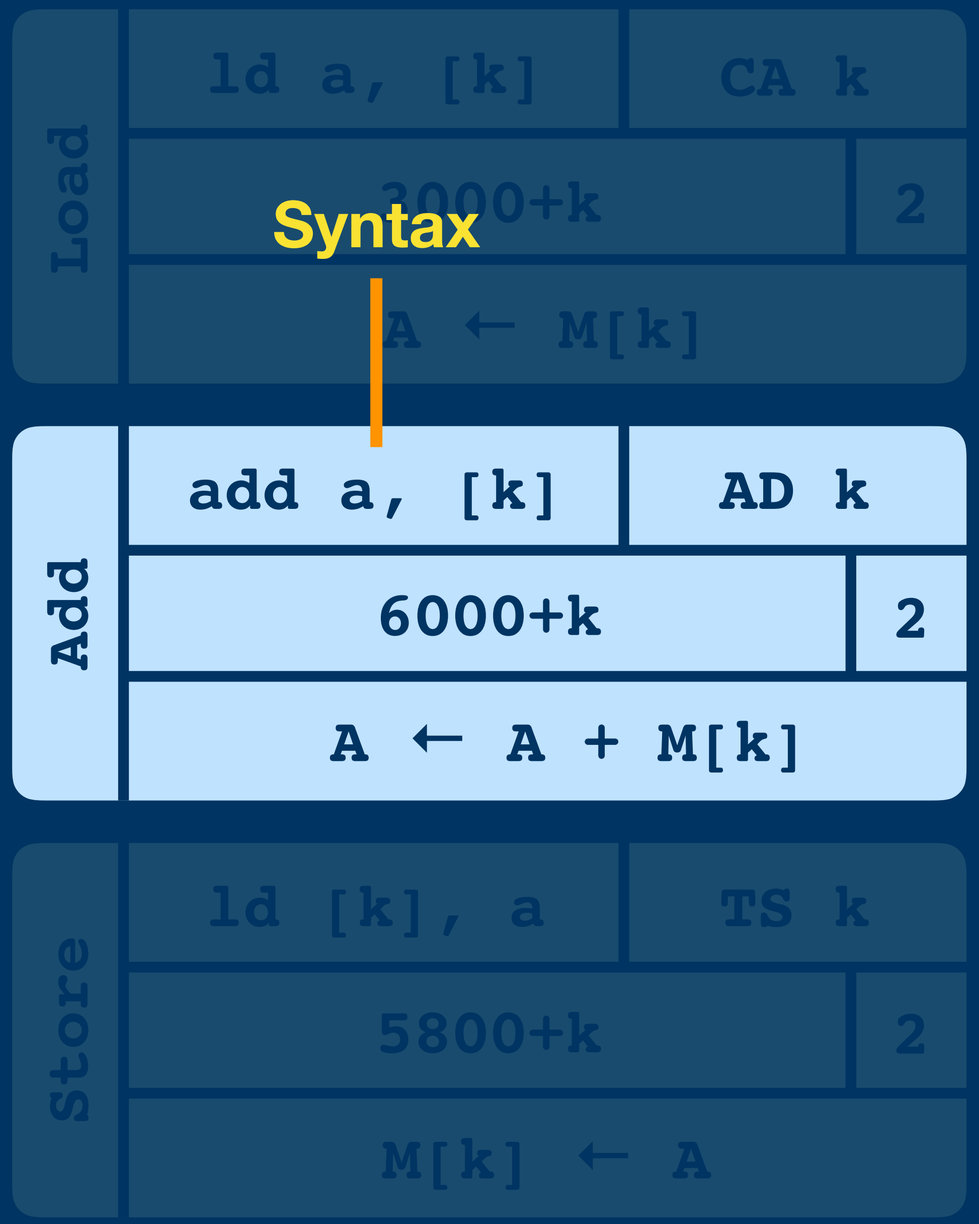
```
ld a, [k]
```

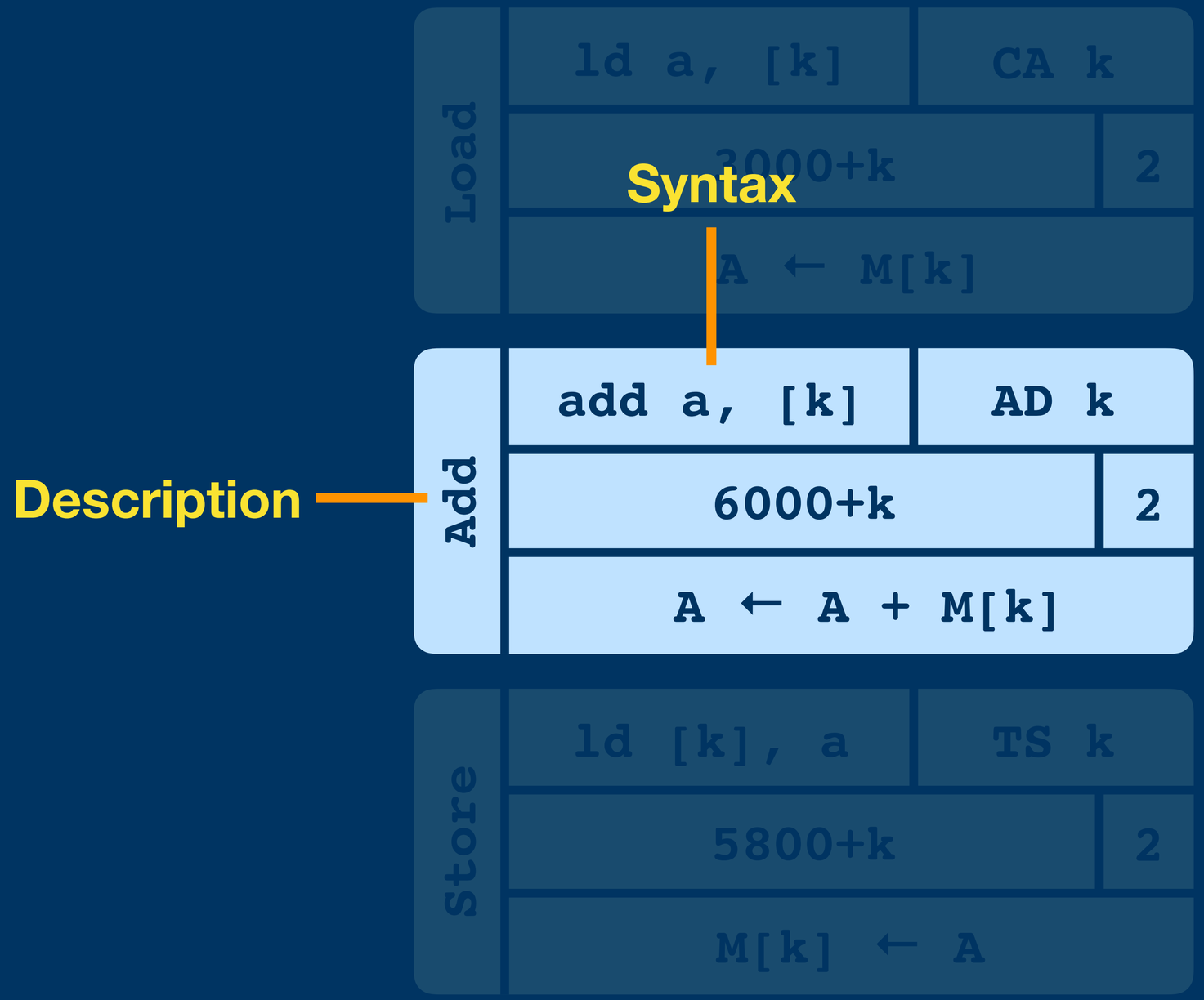
```
add a, [k]
```

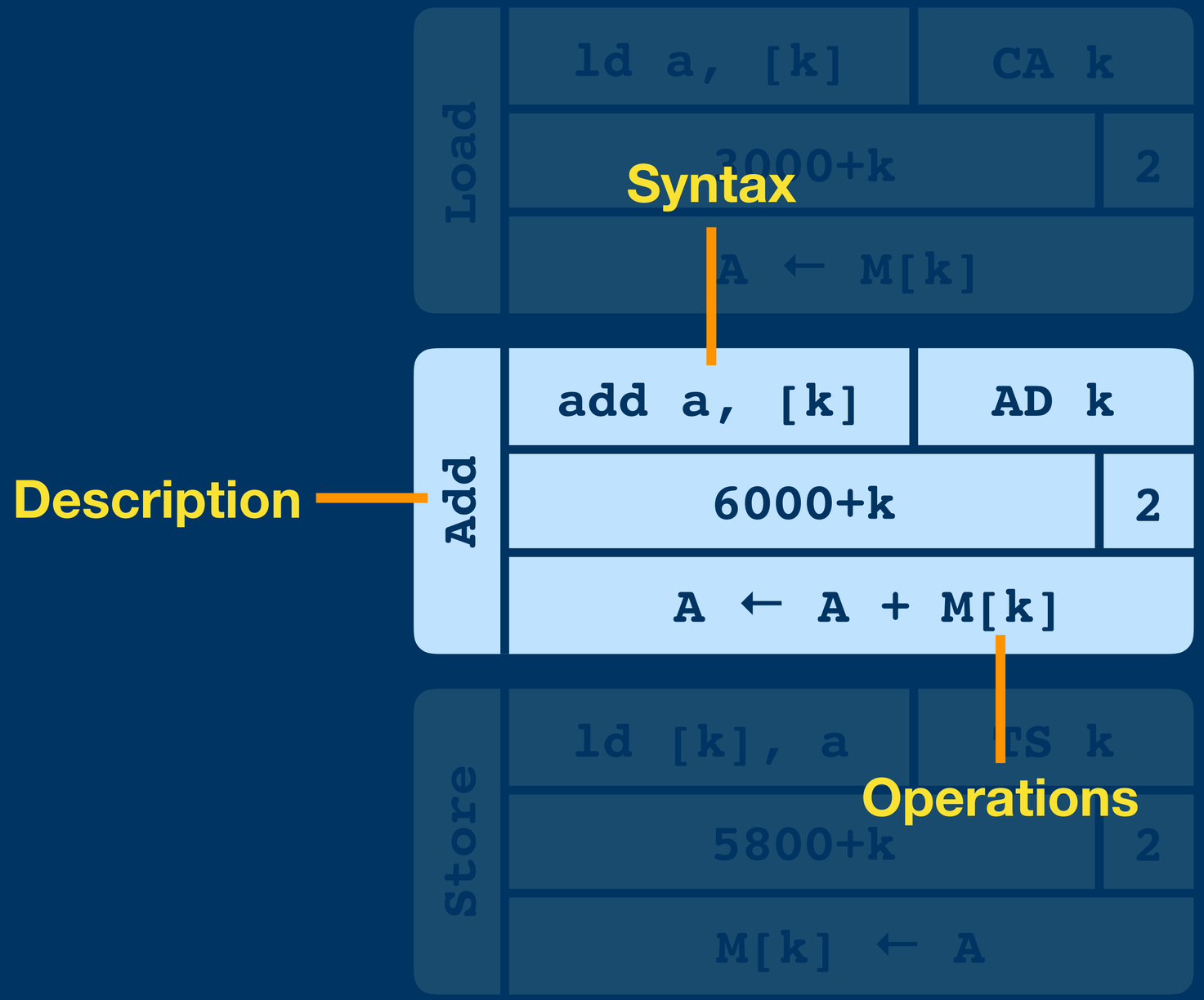
```
ld [k], a
```

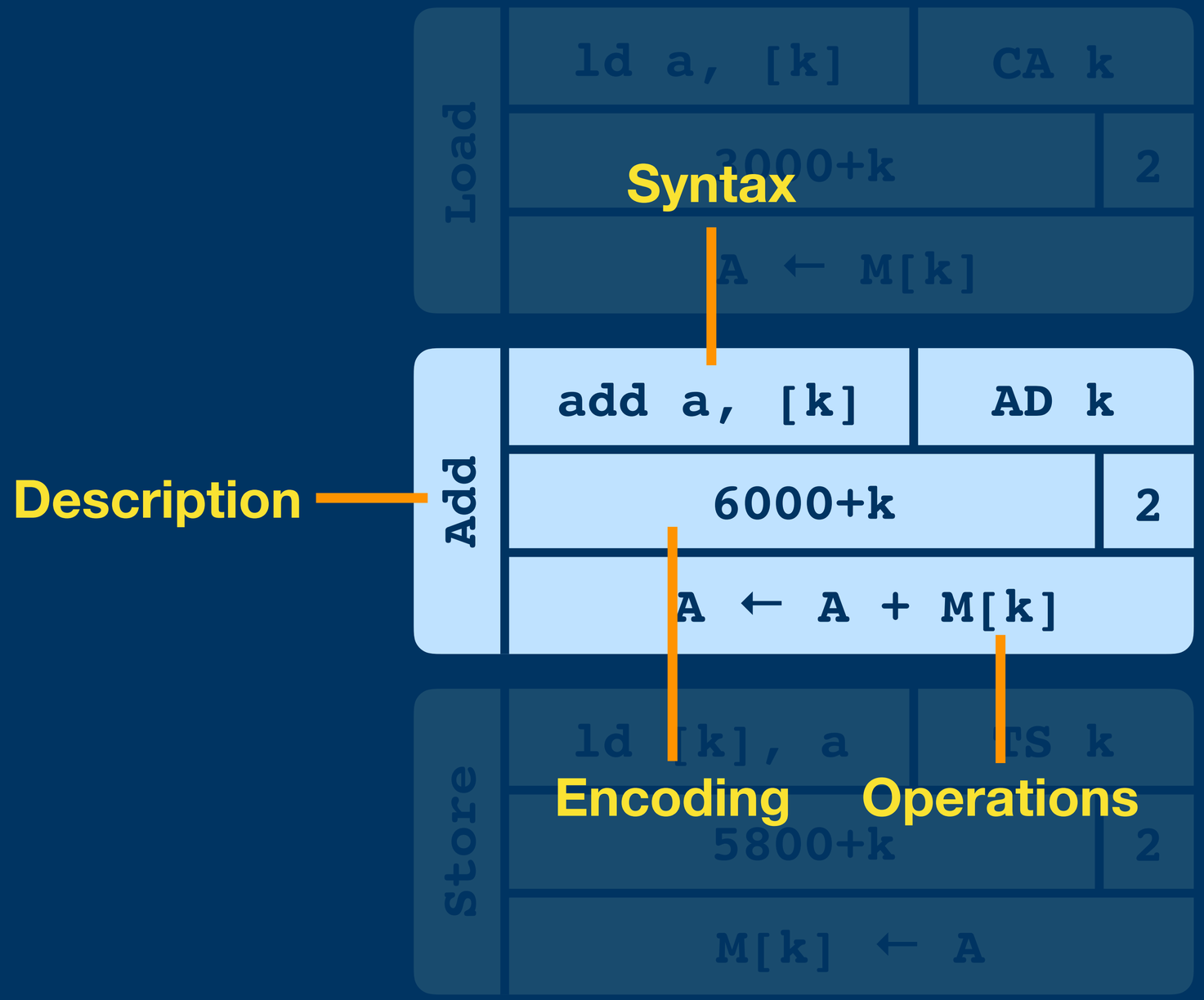

Load
ld a, [k]
CA k
3000+k
2
A ← M[k]
Add
add a, [k]
AD k
6000+k
2
A ← A + M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ← A

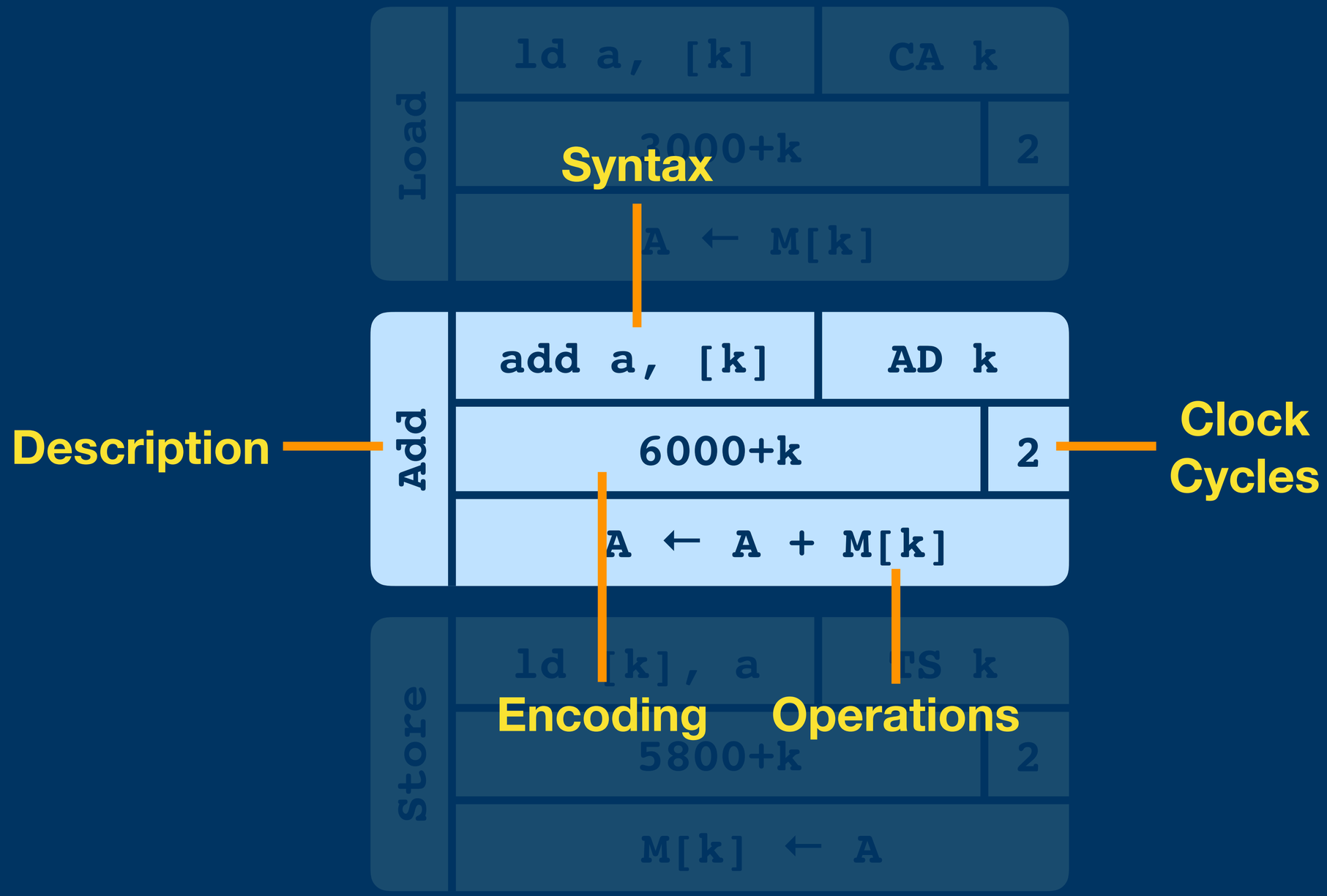

Load
ld a, [k]
CA k
3000+k
2
A ← M[k]
Add
add a, [k]
AD k
6000+k
2
A ← A + M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ← A

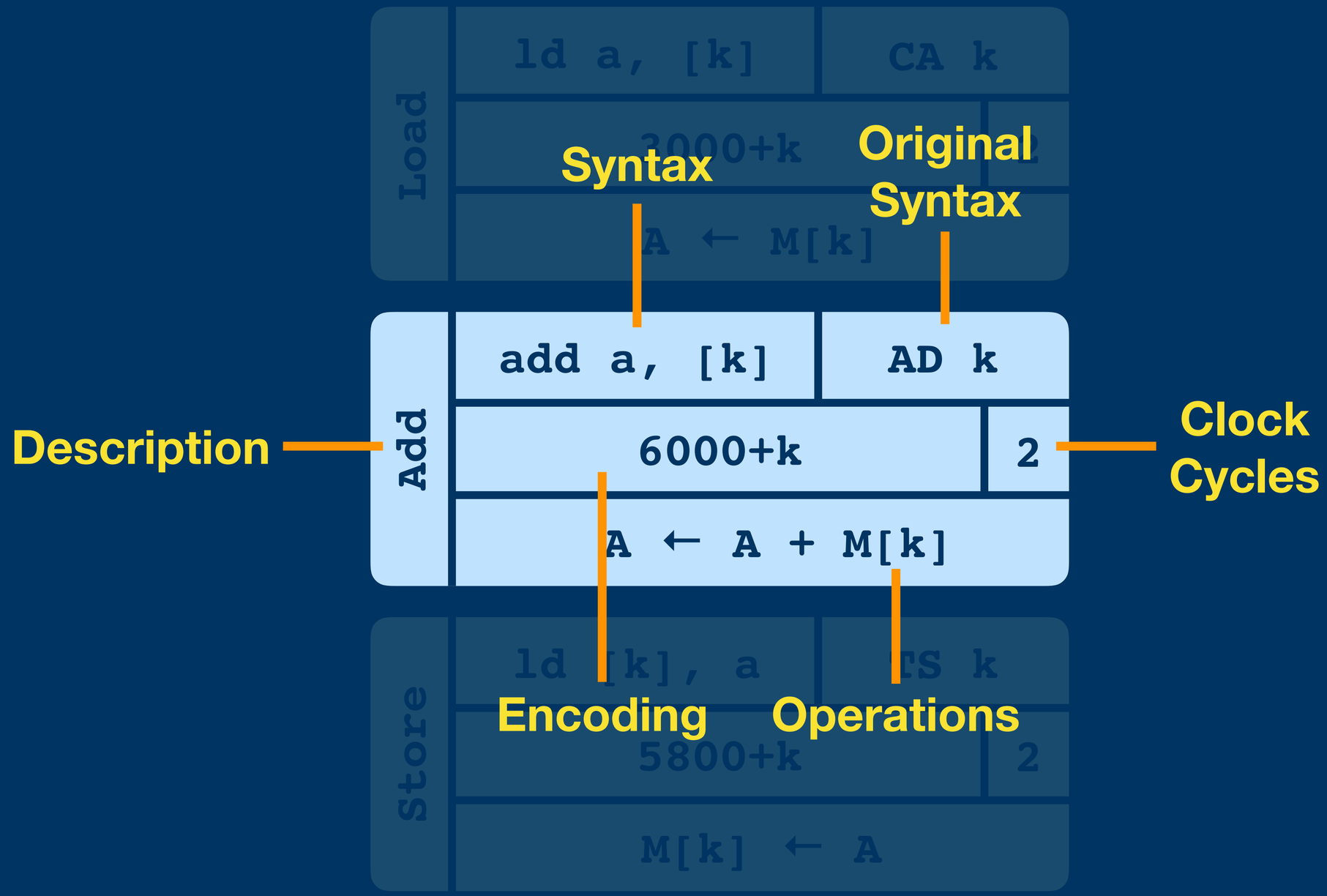

Load
ld a, [k]
CA k
3000+k
2
A ← M[k]
Syntax
Add
add a, [k]
AD k
6000+k
2
A ← A + M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ← A

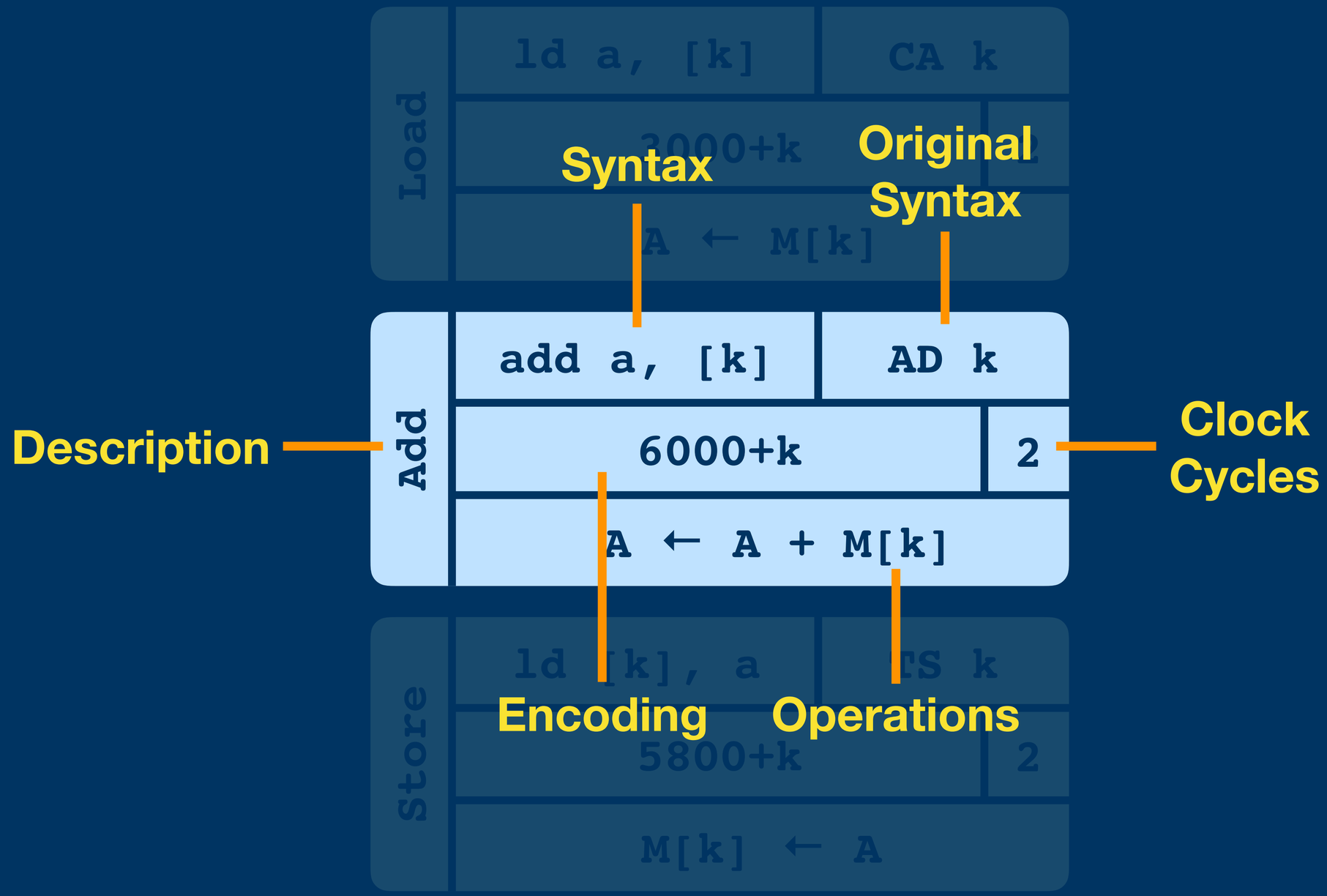

Load
ld a, [k]
CA k
3000+k
2
A ← M[k]
Syntax
Description
Add
add a, [k]
AD k
6000+k
2
A ← A + M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ← A

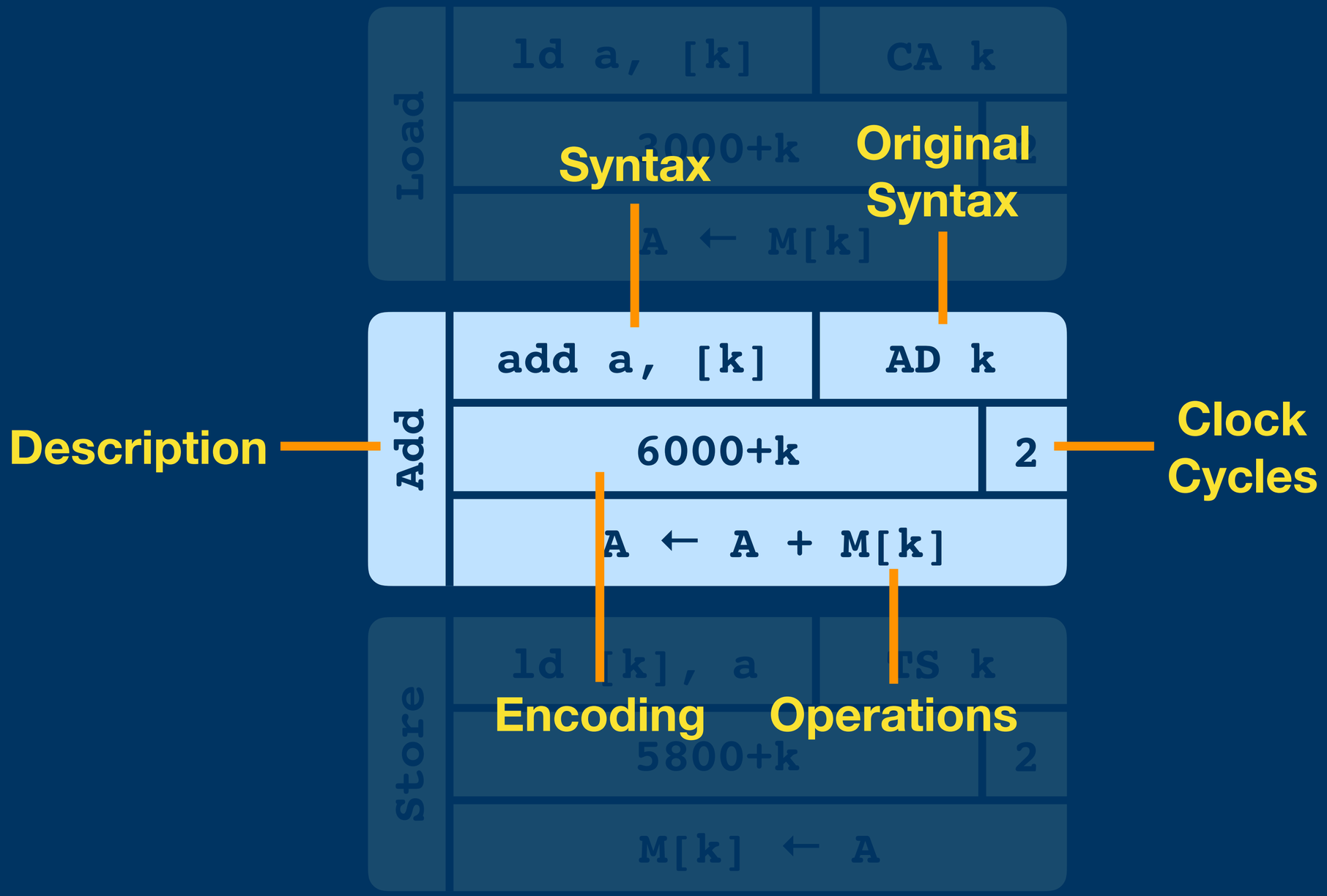

Load
ld a, [k]
CA k
3000+k
2
A ← M[k]
Syntax
Add
add a, [k]
AD k
6000+k
2
A ← A + M[k]
Description
Operations
Store
ld [k], a
TS k
5800+k
2
M[k] ← A


Load
ld a, [k]
CA k
3000+k
2
A ← M[k]
Add
add a, [k]
AD k
6000+k
2
A ← A + M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ← A
Syntax
Description
Encoding
Operations


Load
ld a, [k]
CA k
3000+k
2
A ← M[k]
Add
add a, [k]
AD k
6000+k
2
A ← A + M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ← A
Syntax
Description
Clock Cycles
Encoding
Operations


Load
ld a, [k]
CA k
3000+k
A ← M[k]
Syntax
Original Syntax
Description
Add
add a, [k]
AD k
6000+k
2
A ← A + M[k]
Clock Cycles
Encoding
Operations
Store
ld [k], a
TS k
5800+k
2
M[k] ← A

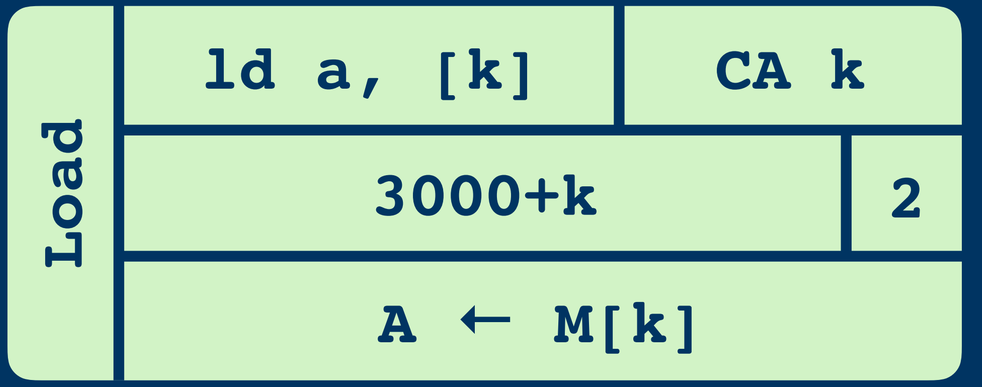

Load
ld a, [k]
CA k
3000+k
2
A ← M[k]
Syntax
Original Syntax
Description
Add
add a, [k]
AD k
6000+k
2
Clock Cycles
A ← A + M[k]
Encoding
Operations
Store
ld [k], a
TS k
5800+k
2
M[k] ← A



Original	*My Syntax*
AD	add a, [k]
ADS	add [k], a
CA	ld a, [k]
CCS	ccs
CS	ldc a, [k]
DAS	add [k], ab
DXCH	xchg ab, [k]
INCR	inc k
INHINT	cli
LXCH	xchg b, [k]
MASK	and a, [k]
RELINT	sti
RESUME	iret
TC	call k
TCF	jmp k
TS	ld [k], a
XCH	xchg a, [k]

Load

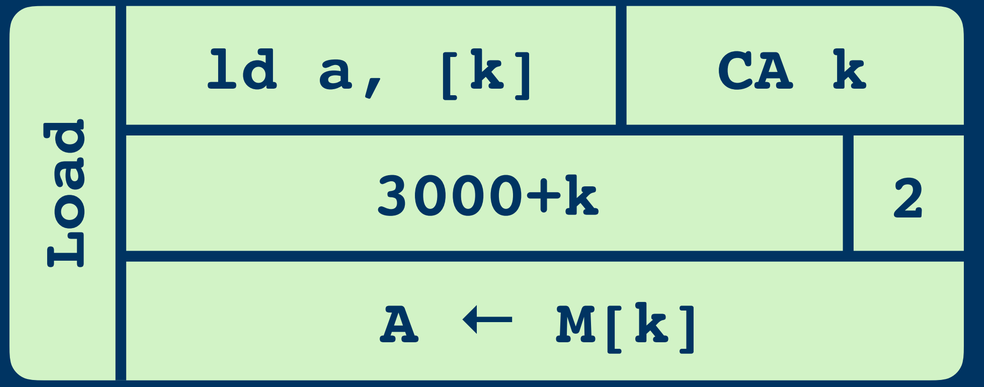
```
ld a, [$200]
```



Load

```
ld a, [$200]
```

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000



Load

ld a, [$200]

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

281F	1FC
0CDC	1FD
3193	1FE
4584	1FF
1111	200
0A0A	201
37B5	202
52A3	203
24D1	204

Load		
	ld a, [k]	CA k
	3000+k	2
	A ← M[k]	

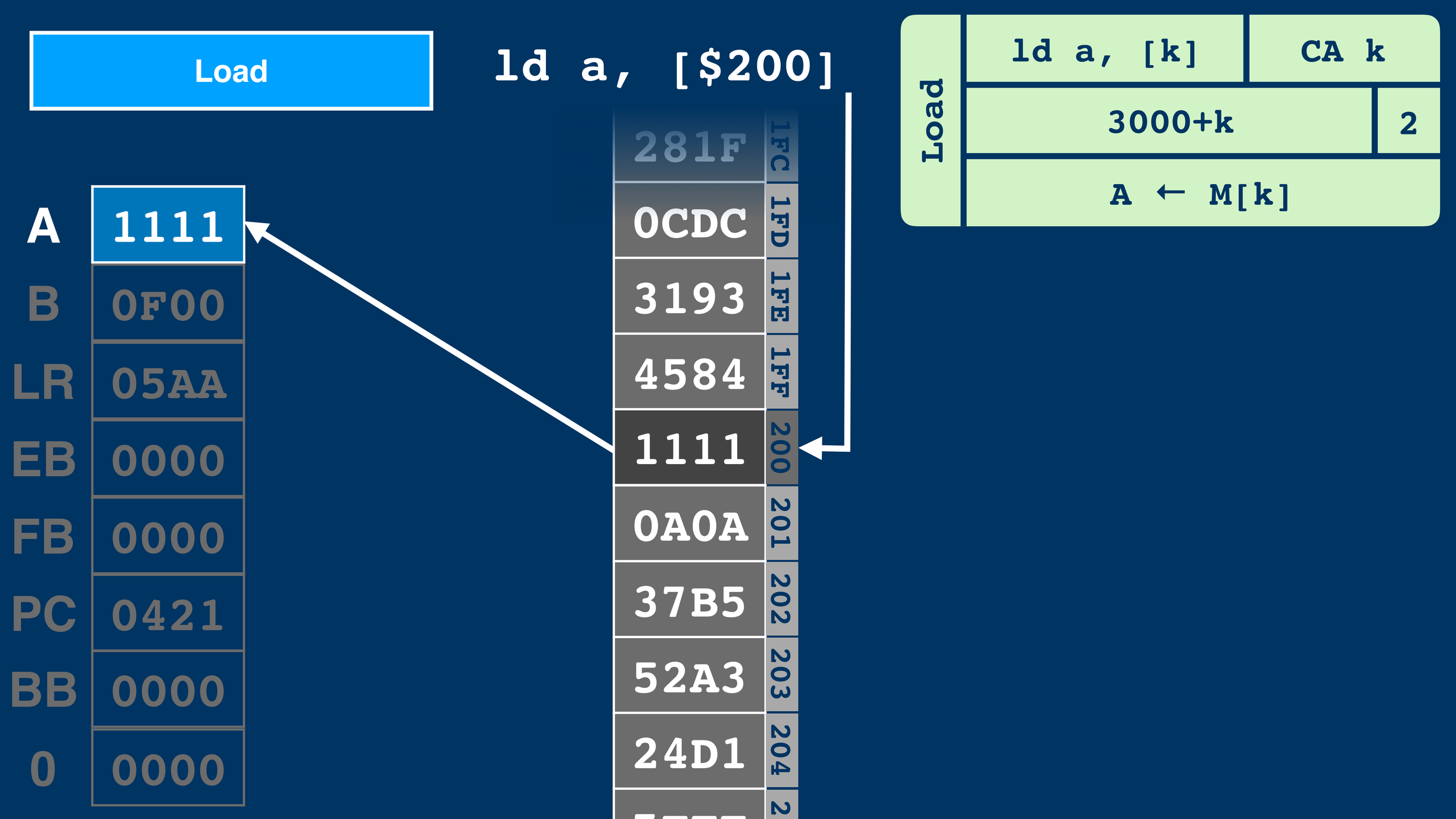
Load

```
ld a, [$200]
```

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

1FC	281F
1FD	0CDC
1FE	3193
1FF	4584
200	1111
201	0A0A
202	37B5
203	52A3
204	24D1

Load		
	ld a, [k]	CA k
	3000+k	2
	A ← M[k]	


Load
ld a, [$200]
Load
ld a, [k]
CA k
3000+k
2
A ← M[k]
A 1111
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
281F 1FC
0CDC 1FD
3193 1FE
4584 1FF
1111 200
0A0A 201
37B5 202
52A3 203
24D1 204

Store

```
ld [$200], a
```

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Store		
	ld [k], a	TS k
	5800+k	2
	M[k] ← A	

Store

ld [$200], a

Store	ld [k], a	TS k
	5800+k	2
	M[k] ← A	

Exchange

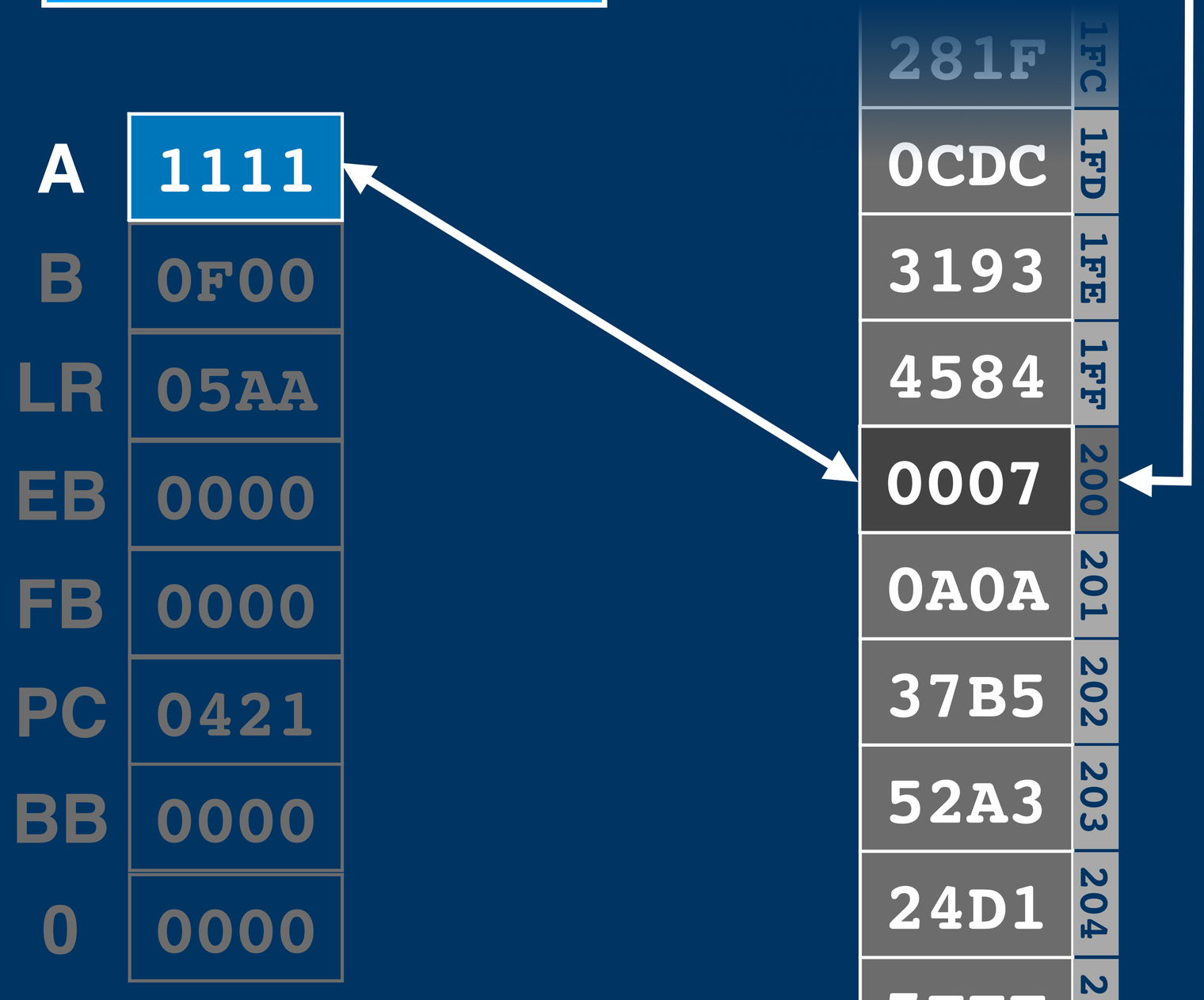
```
xchg a, [$200]
```

Exchange		
	xchg a, [k]	XCH k
	5C00+k	2
	M[k] ↔ A	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Exchange

```
xchg a, [$200]
```



Exchange	xchg a, [k]	XCH k
	5C00+k	2
	M[k] ↔ A	

Add

```
add a, [$200]
```

Add	add a, [k]	AD k
	6000+k	2
	A ← A + M[k]	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Add

`add a, [$200]`

A	1118
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

1FC	281F	
1FD	0CDC	
1FE	3193	
1FF	4584	
200	1111	←
201	0A0A	
202	37B5	
203	52A3	
204	24D1	

Add		
	add a, [k]	AD k
	6000+k	2
	A ← A + M[k]	

Subtract

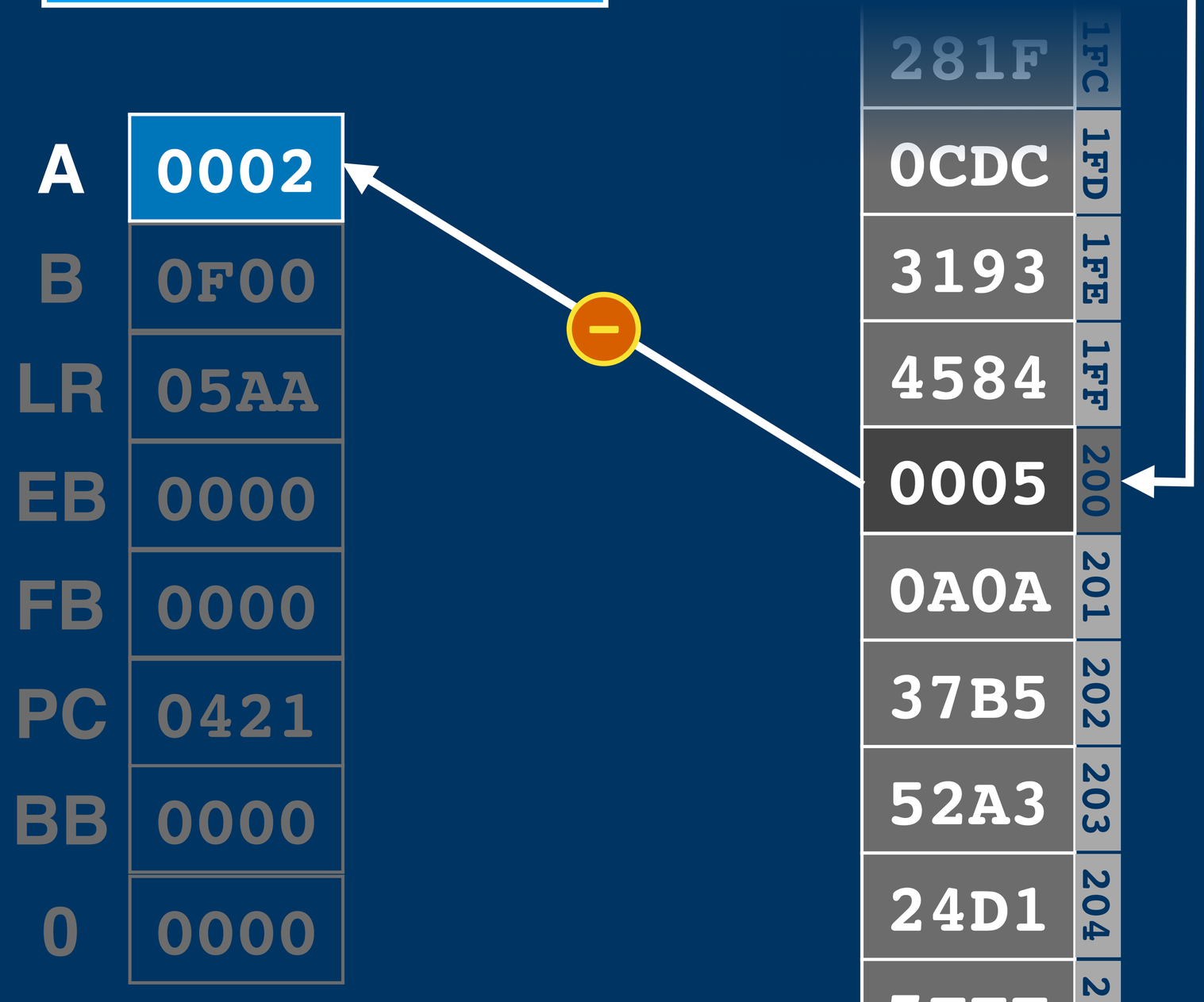
```
sub a, [$200]
```

Subtract		
	sub a, [k]	SU k
	0006, 6000+k	3
	A ← A - M[k]	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Subtract

```
sub a, [$200]
```

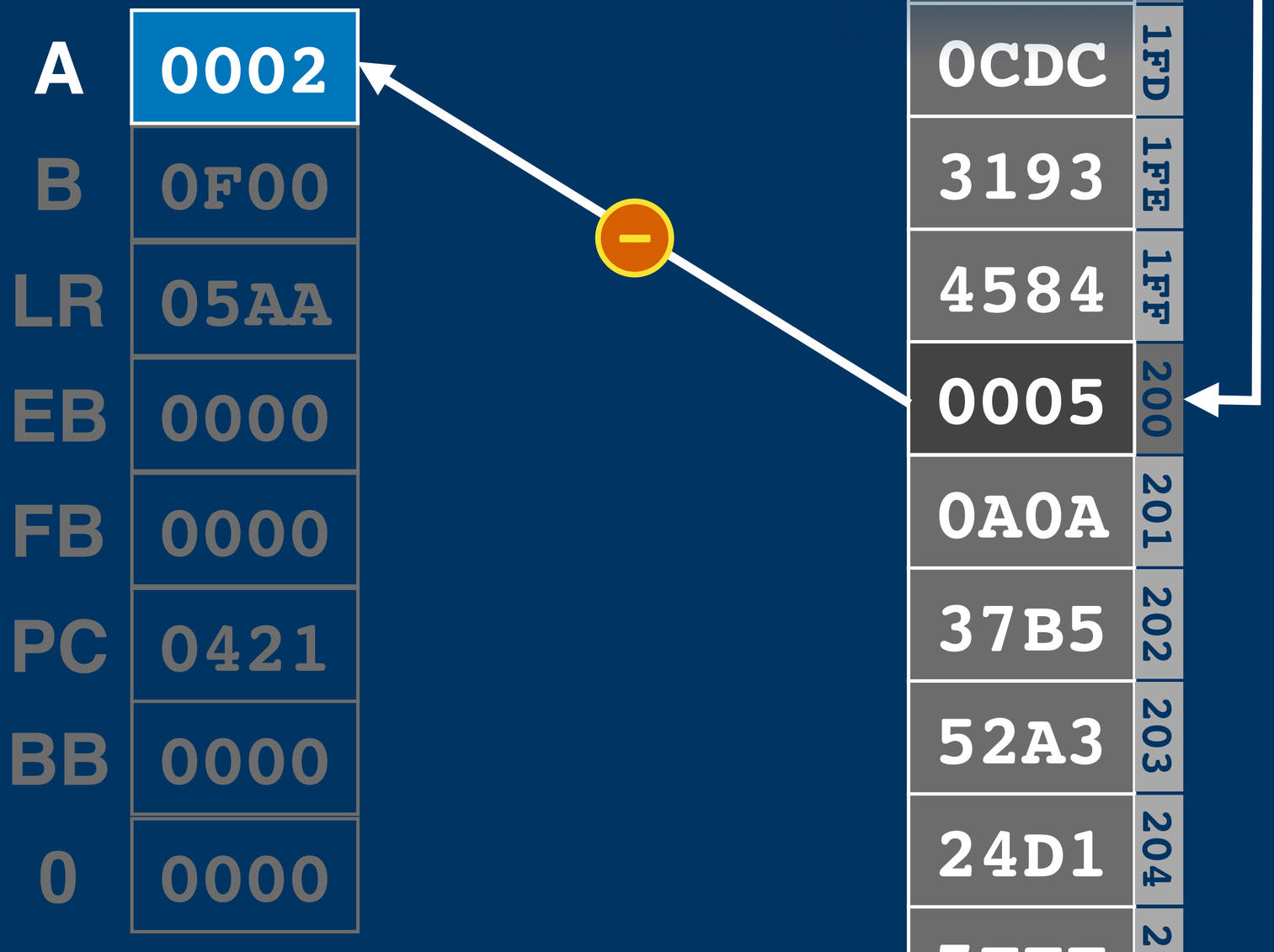


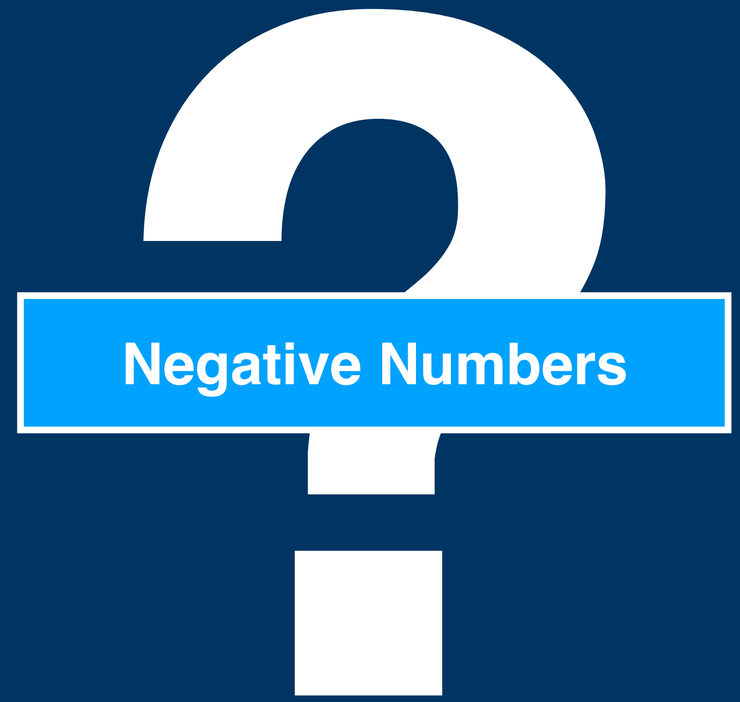
Subtract		
sub a, [k]		SU k
0006, 6000+k		3
A ← A - M[k]		

Subtract

```
sub a, [$200]
```

Subtract		
	sub a, [k]	SU k
	0006, 6000+k	3
	A ← A - M[k]	





Negative Numbers

Negative Numbers

Unsigned

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Negative Numbers

Unsigned		Sign-and-value	
0000	0	0000	+0
0001	1	0001	+1
0010	2	0010	+2
0011	3	0011	+3
0100	4	0100	+4
0101	5	0101	+5
0110	6	0110	+6
0111	7	0111	+7
1000	8	1000	−0
1001	9	1001	−1
1010	10	1010	−2
1011	11	1011	−3
1100	12	1100	−4
1101	13	1101	−5
1110	14	1110	−6
1111	15	1111	−7

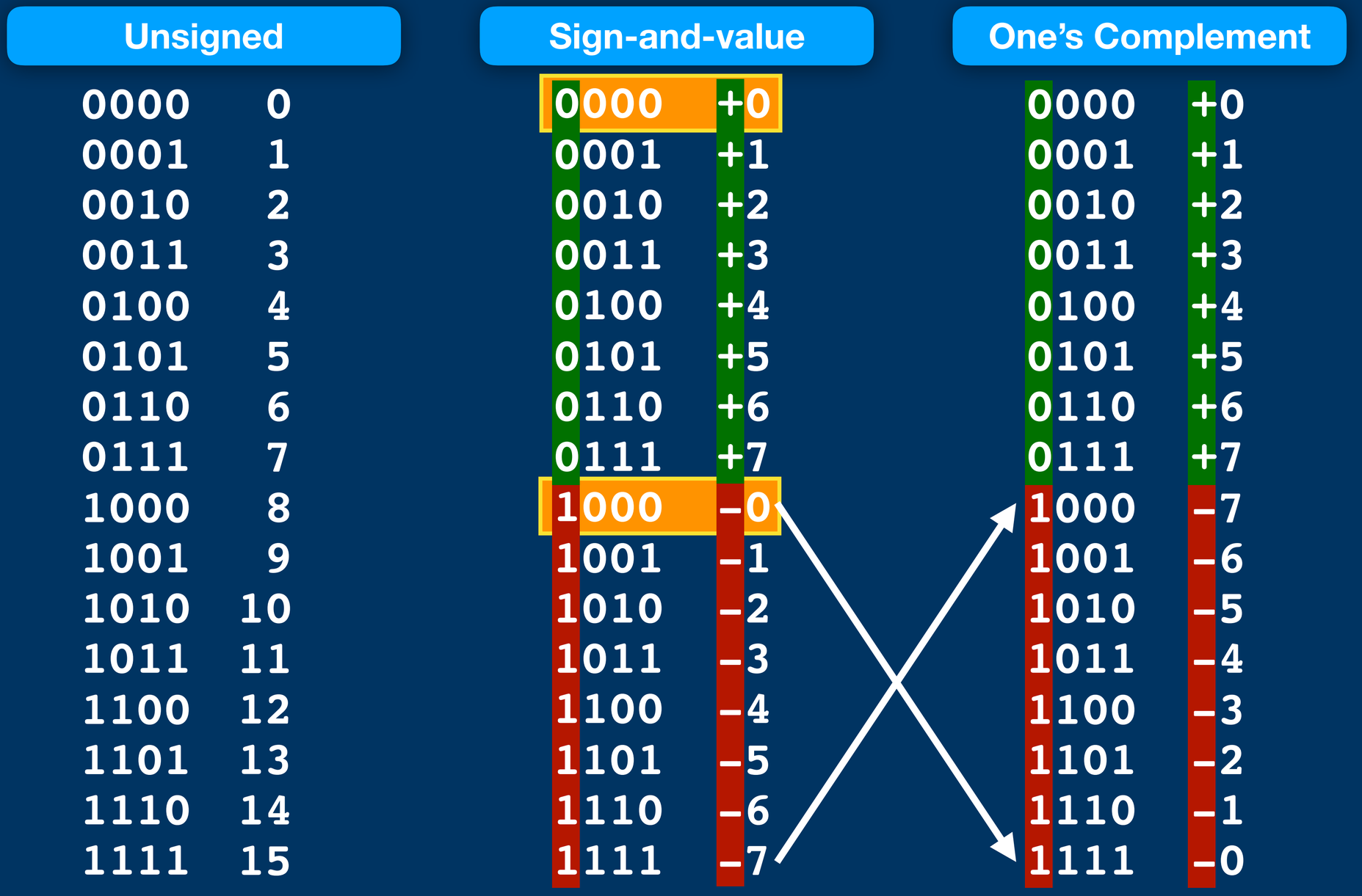
Negative Numbers

Unsigned		Sign-and-value	
0000	0	0000	+0
0001	1	0001	+1
0010	2	0010	+2
0011	3	0011	+3
0100	4	0100	+4
0101	5	0101	+5
0110	6	0110	+6
0111	7	0111	+7
1000	8	1000	−0
1001	9	1001	−1
1010	10	1010	−2
1011	11	1011	−3
1100	12	1100	−4
1101	13	1101	−5
1110	14	1110	−6
1111	15	1111	−7

Negative Numbers

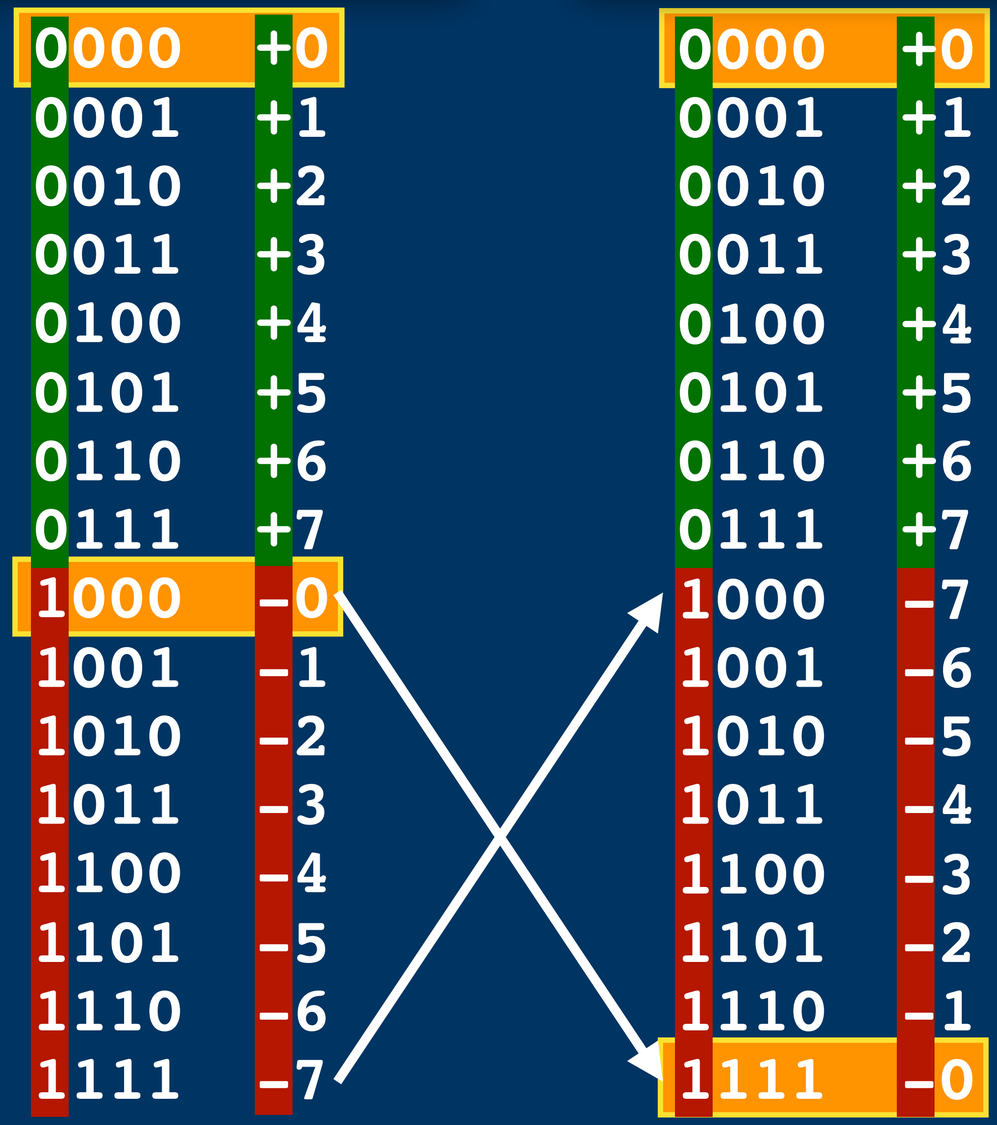
Unsigned		Sign-and-value	
0000	0	0000	+0
0001	1	0001	+1
0010	2	0010	+2
0011	3	0011	+3
0100	4	0100	+4
0101	5	0101	+5
0110	6	0110	+6
0111	7	0111	+7
1000	8	1000	−0
1001	9	1001	−1
1010	10	1010	−2
1011	11	1011	−3
1100	12	1100	−4
1101	13	1101	−5
1110	14	1110	−6
1111	15	1111	−7

Negative Numbers

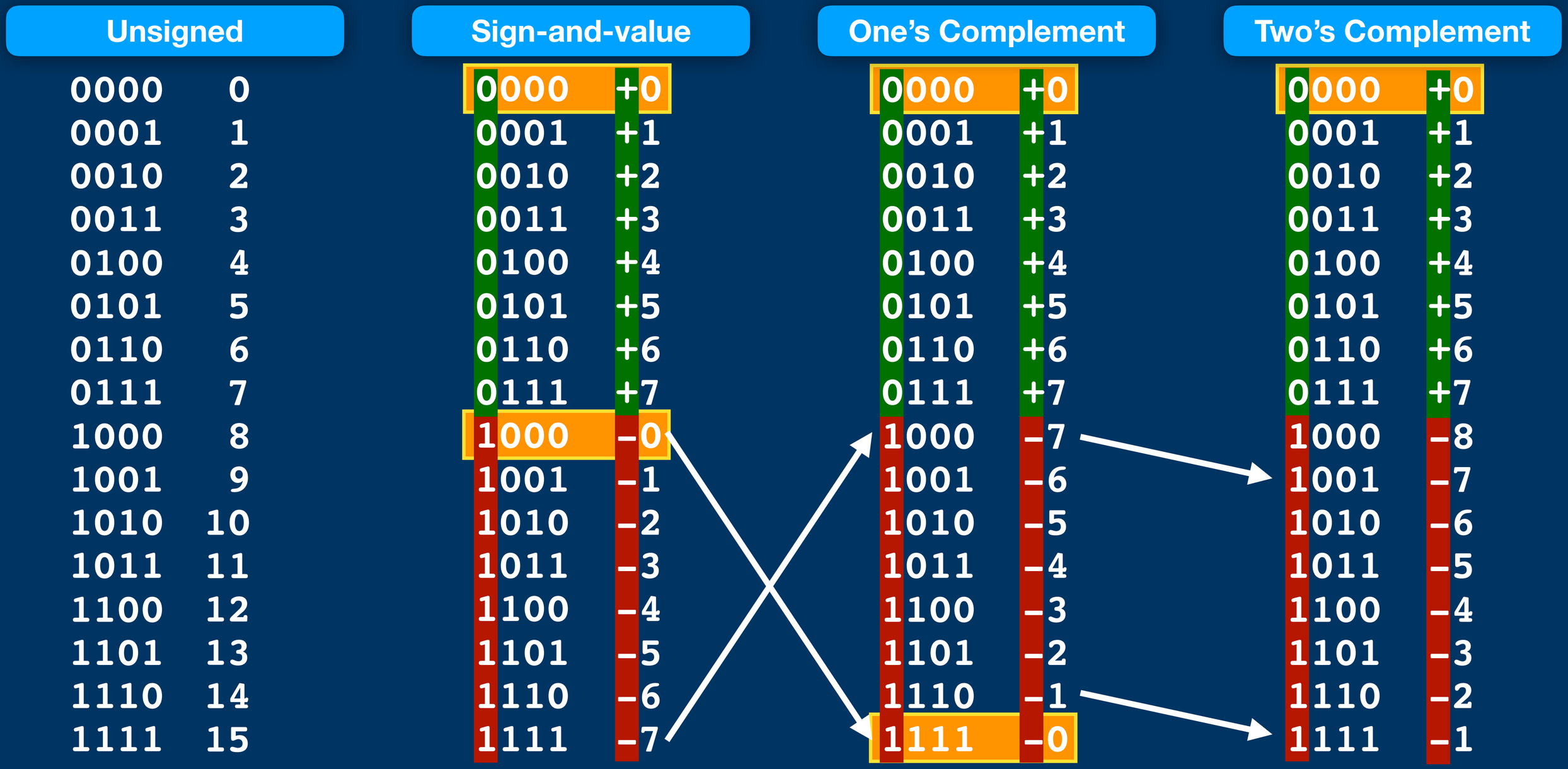


Negative Numbers

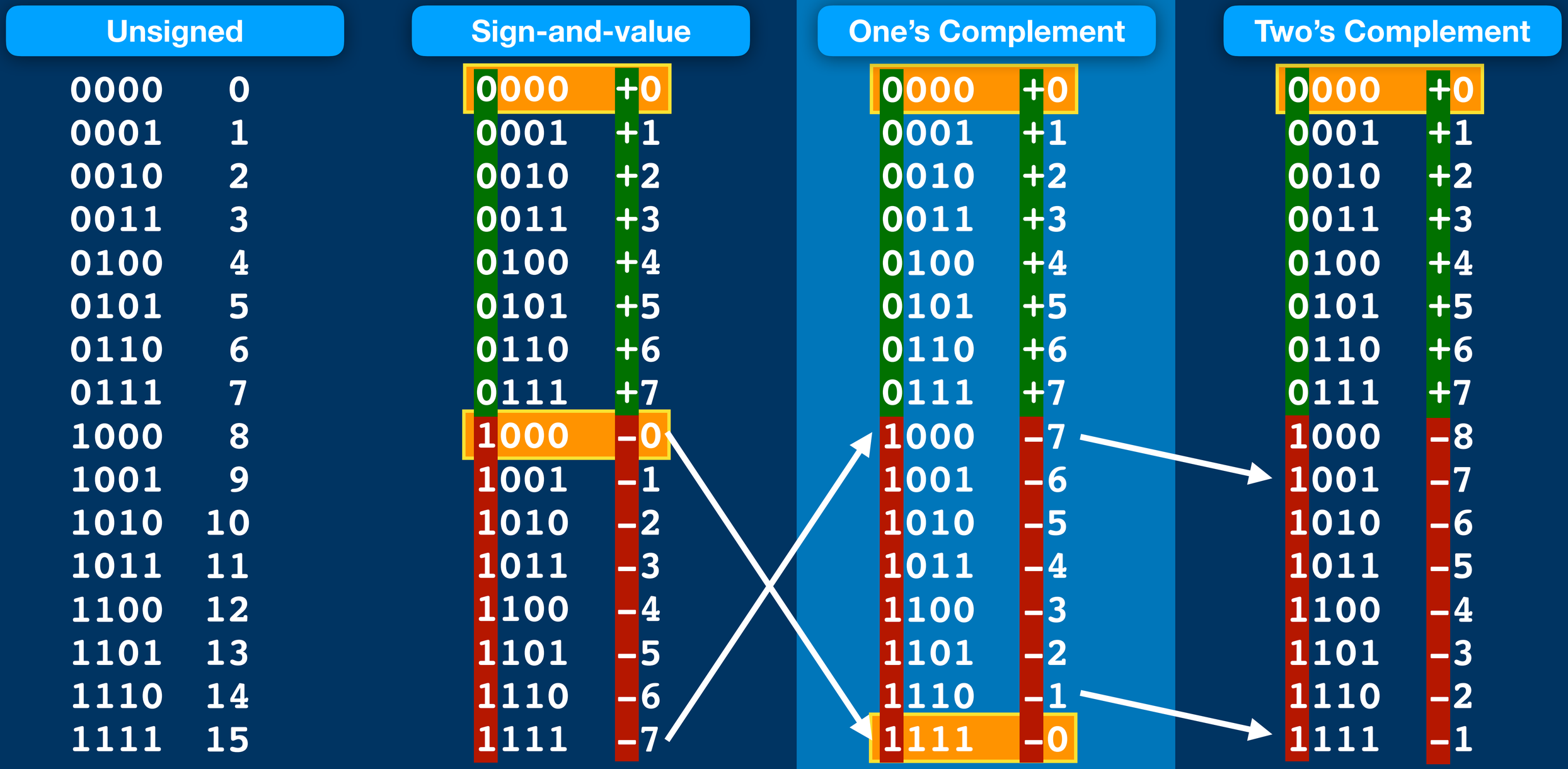
Unsigned		Sign-and-value		One's Complement	
0000	0	0000	+0	0000	+0
0001	1	0001	+1	0001	+1
0010	2	0010	+2	0010	+2
0011	3	0011	+3	0011	+3
0100	4	0100	+4	0100	+4
0101	5	0101	+5	0101	+5
0110	6	0110	+6	0110	+6
0111	7	0111	+7	0111	+7
1000	8	1000	−0	1000	−7
1001	9	1001	−1	1001	−6
1010	10	1010	−2	1010	−5
1011	11	1011	−3	1011	−4
1100	12	1100	−4	1100	−3
1101	13	1101	−5	1101	−2
1110	14	1110	−6	1110	−1
1111	15	1111	−7	1111	−0



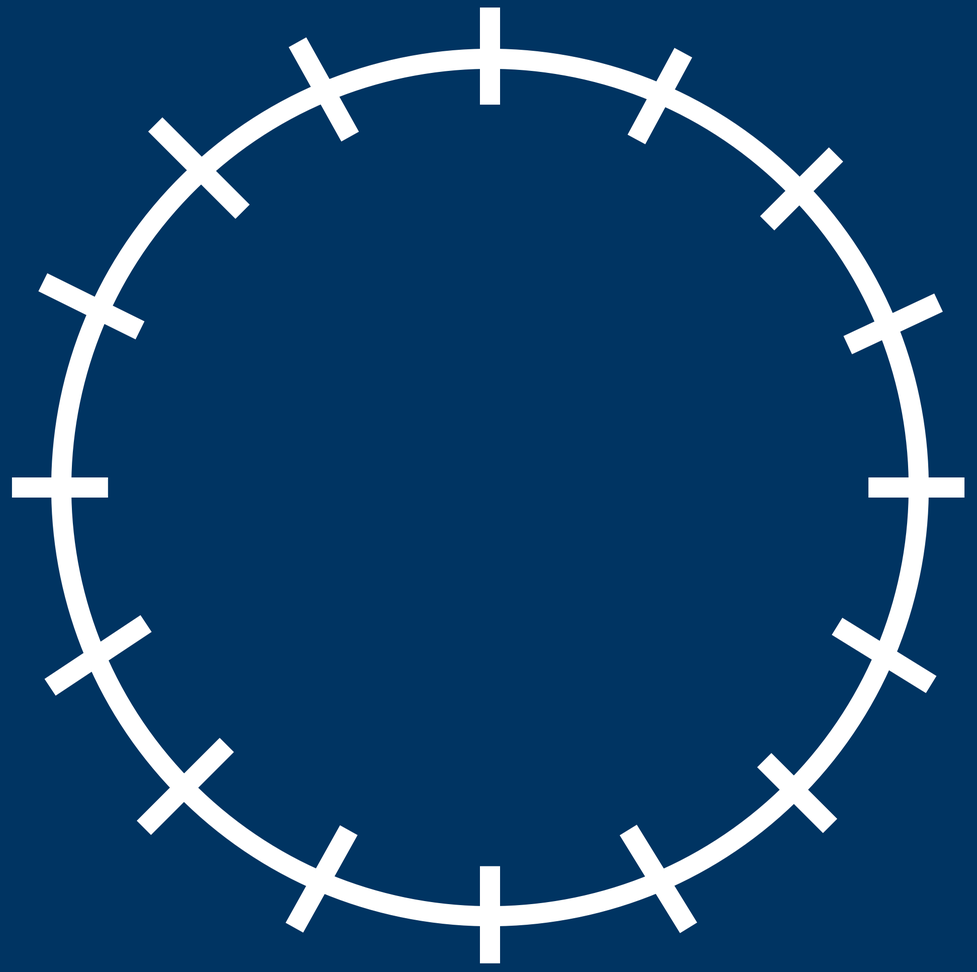
Negative Numbers



Negative Numbers

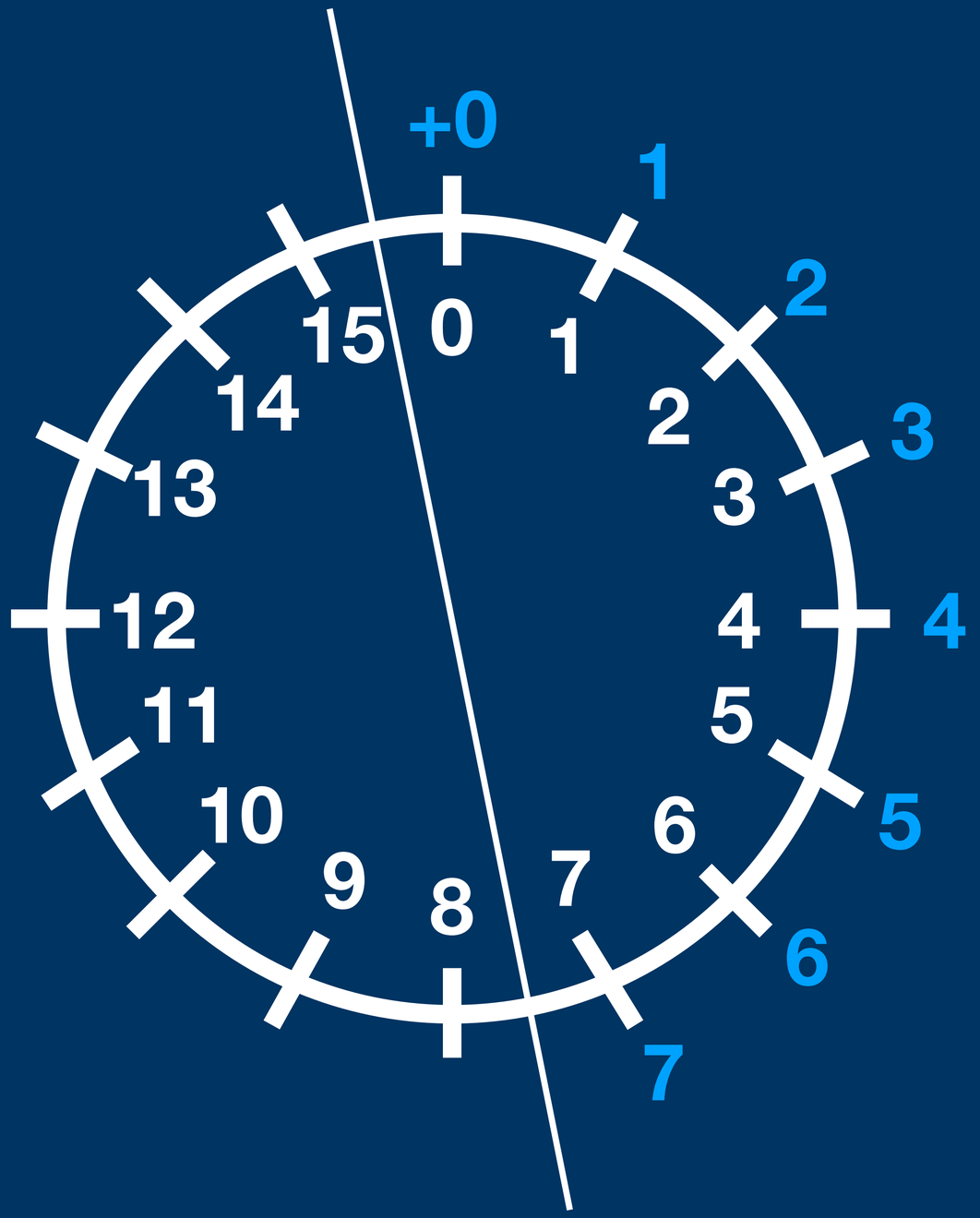


One’s Complement



One’s Complement

One’s Complement



One’s Complement

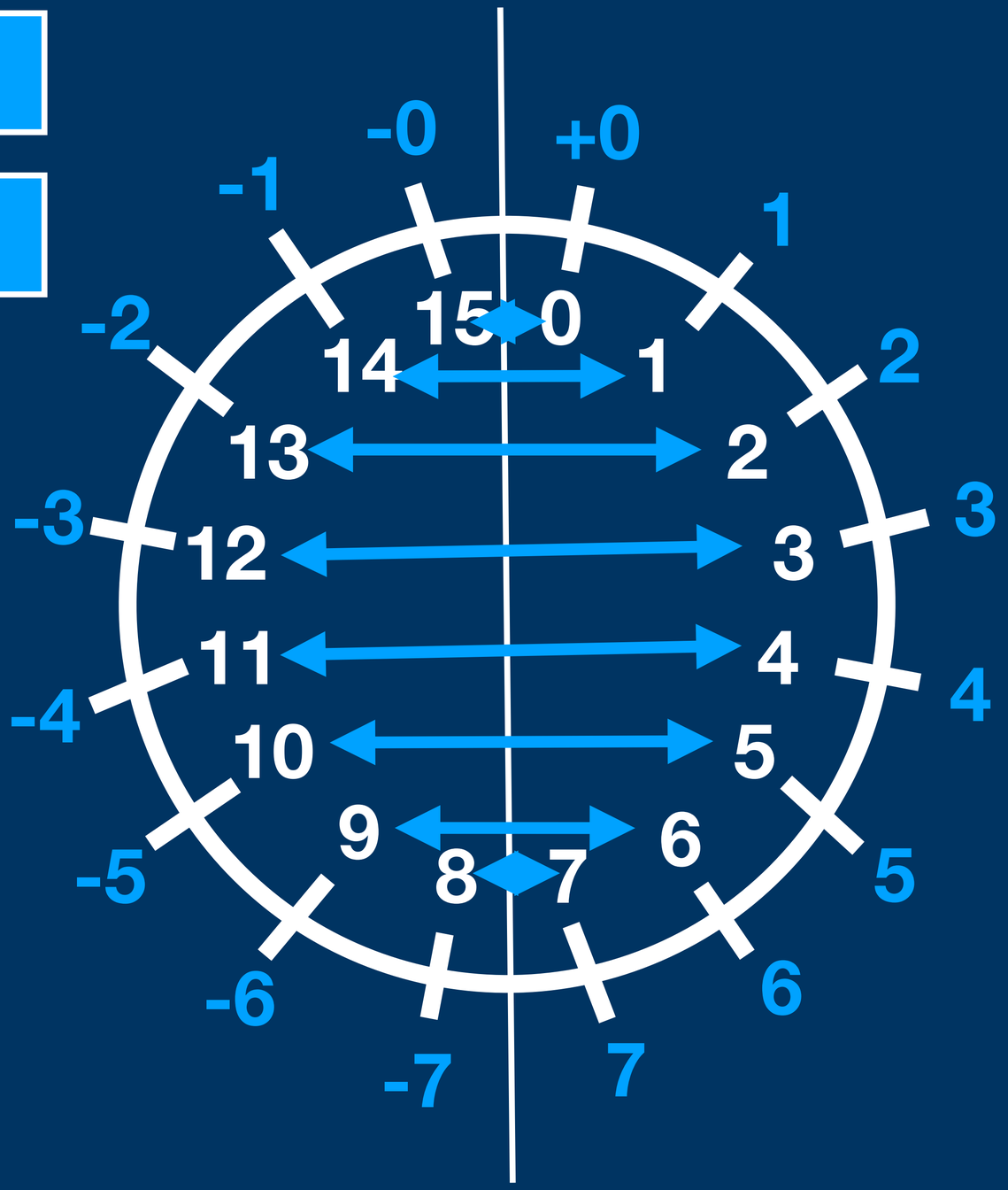
One’s Complement

One’s Complement

Negate

One’s Complement

Negate



One’s Complement

Negate

+0	0000	↔	1111	-0
+1	0001	↔	1110	-1
+2	0010	↔	1101	-2
+3	0011	↔	1100	-3
+4	0100	↔	1011	-4
+5	0101	↔	1010	-5
+6	0110	↔	1011	-6
+7	0111	↔	1000	-7

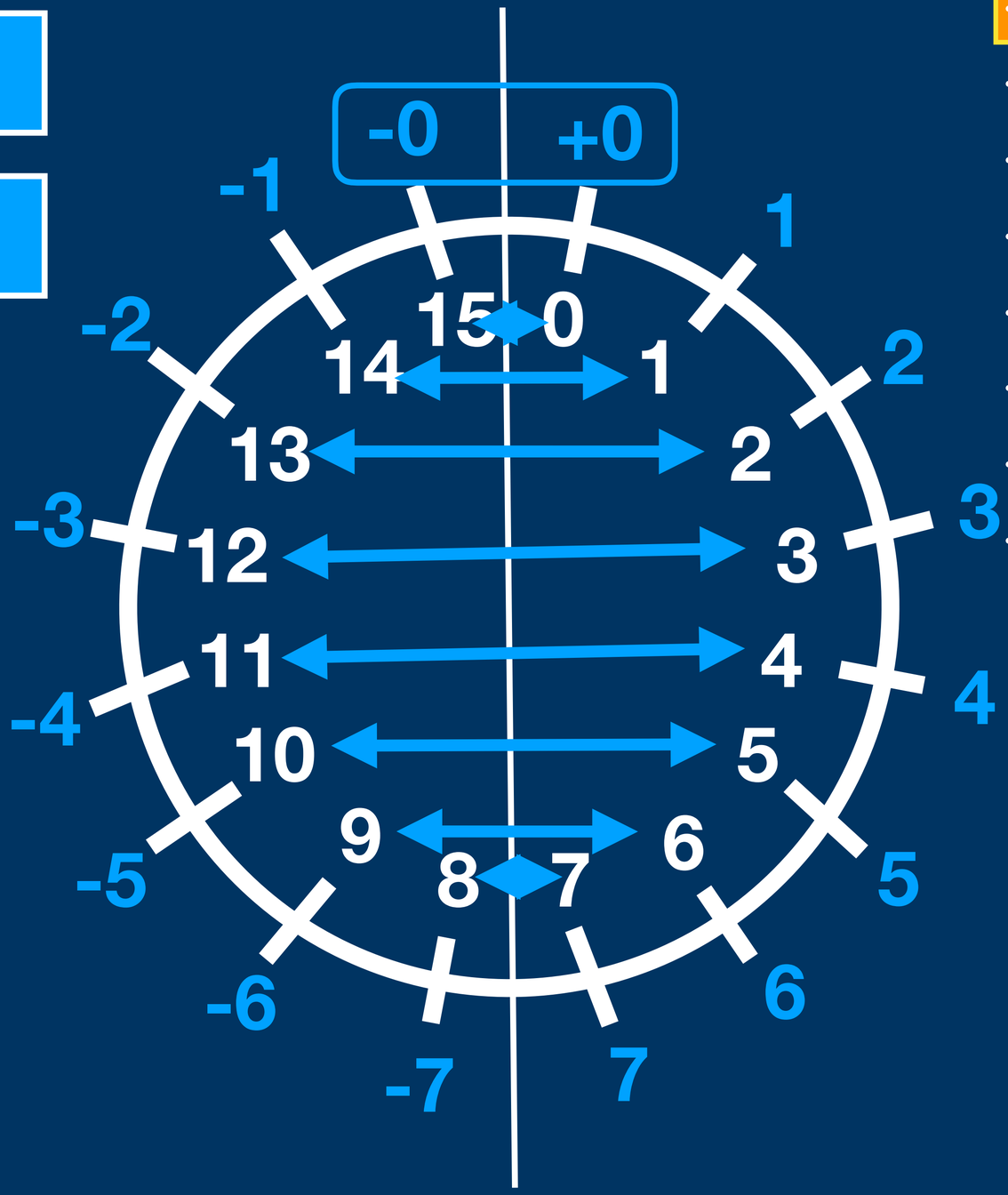
One’s Complement

Negate

+0	0000	↔	1111	-0
+1	0001	↔	1110	-1
+2	0010	↔	1101	-2
+3	0011	↔	1100	-3
+4	0100	↔	1011	-4
+5	0101	↔	1010	-5
+6	0110	↔	1011	-6
+7	0111	↔	1000	-7

One’s Complement

Negate



+0	0000	↔	1111	-0
+1	0001	↔	1110	-1
+2	0010	↔	1101	-2
+3	0011	↔	1100	-3
+4	0100	↔	1011	-4
+5	0101	↔	1010	-5
+6	0110	↔	1011	-6
+7	0111	↔	1000	-7

One’s Complement

Add

One’s Complement

Add

1 + 2

= 1 + 2

= 3

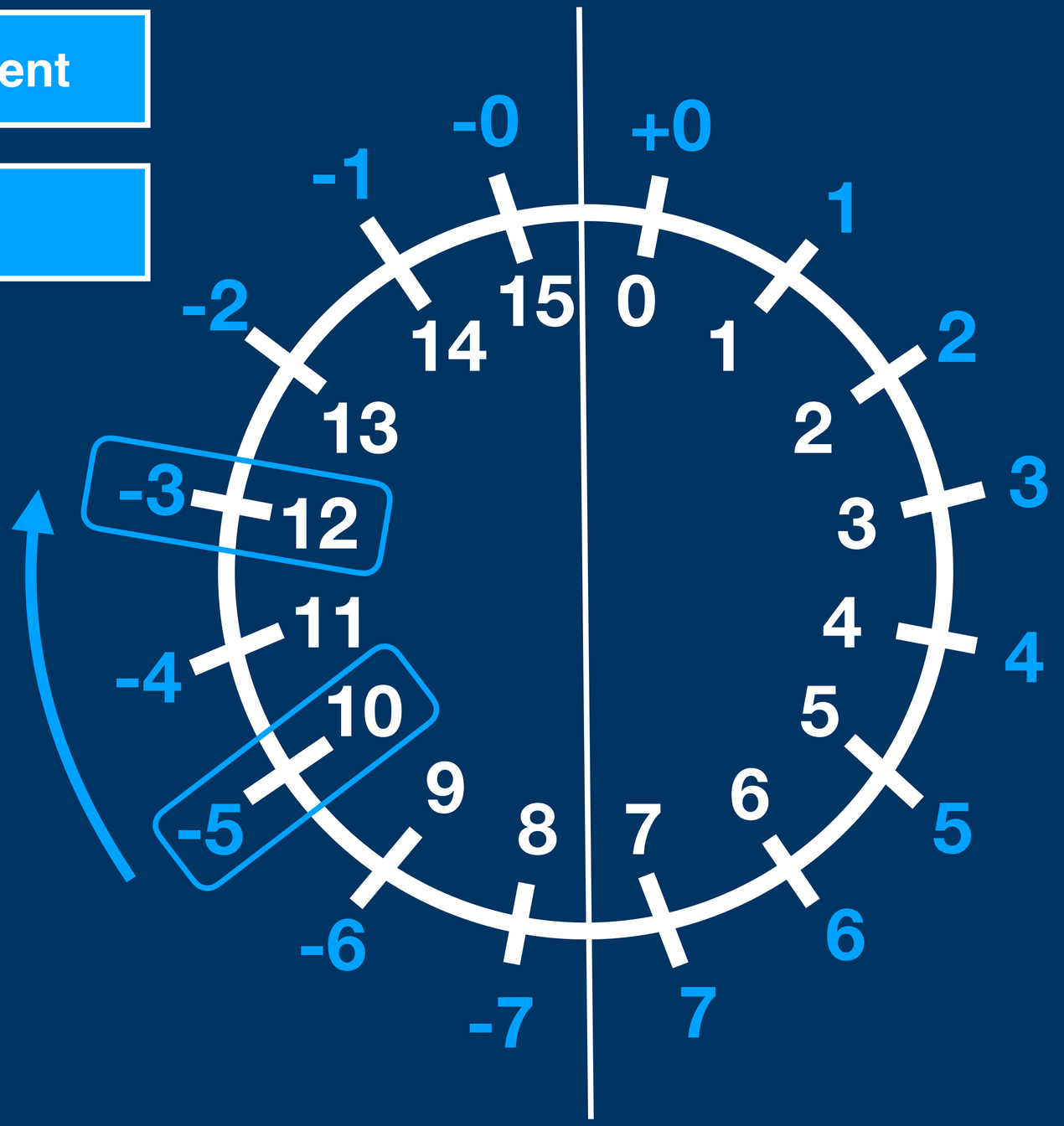
= 3

One’s Complement

Add

One’s Complement

Add



$$
\begin{aligned}
&-5 + 2 \\
&= 10 + 2 \\
&= 12 \\
&= -3
\end{aligned}
$$

One’s Complement

Subtract

One’s Complement

Subtract

6 – 4

One’s Complement

Subtract

$$6 - 4$$
$$= 6 + (-4)$$

One’s Complement

Subtract

$$
\begin{aligned}
& 6 - 4 \\
= \; & 6 + (-4) \\
= \; & 6 + 11
\end{aligned}
$$

One’s Complement

Subtract


```
  6 - 4
= 6 + (-4)
= 6 + 11
```

One’s Complement

Subtract

$$
\begin{aligned}
& 6 - 4 \\
= \; & 6 + (-4) \\
= \; & 6 + 11 \\
= \; & 1_{\text{Carry}}
\end{aligned}
$$

One’s Complement

Subtract

$$
\begin{aligned}
& 6 - 4 \\
= \; & 6 + (-4) \\
= \; & 6 + 11 \\
= \; & 1_{\text{Carry}} \\
= \; & 2
\end{aligned}
$$

One’s Complement

Subtract

$$
\begin{aligned}
& 6 - 4 \\
= \; & 6 + (-4) \\
= \; & 6 + 11 \\
= \; & 1\,\text{Carry} \\
= \; & 2
\end{aligned}
$$

End-Around-Carry

One’s Complement

Overflow

One’s Complement

Overflow


```
    7 + 1
=   7 + 1
=   8
= -7
```

Positive Overflow

One’s Complement

One’s Complement

$$
\begin{aligned}
& -7 - 1 \\
= \; & -7 + (-1) \\
= \; & 8 + 14 \\
= \; & 6 \quad \text{Carry} \\
= \; & 7
\end{aligned}
$$

Negative Overflow

One’s Complement

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000


```
  -7 - 1
= -7 + (-1)
=  8 + 14
=  6   Carry
=  7
```

Negative
Overflow

One's Complement

A	0007 (oVerflow)
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000


```
   -7 - 1
= -7 + (-1)
=  8 + 14
=  6 Carry
=  7
```

One's Complement

A	0007	V
B	0F00	
LR	05AA	
EB	0000	
FB	0000	
PC	0421	
BB	0000	
0	0000	


```
  -7 - 1
= -7 + (-1)
=  8 + 14
=  6 Carry
=  7
```

Overflow Handling

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

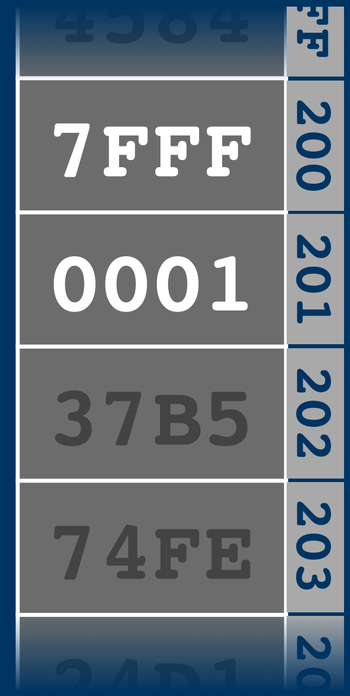
```
ld a, [$200]
add a, [$201]
ld [$203], a
jmp cont
ld [$202], a
```

Overflow Handling

A	7FFF
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

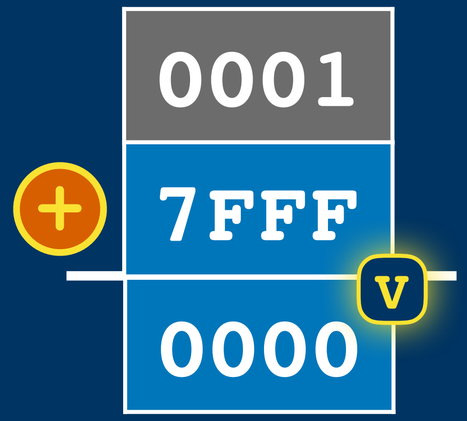
PC →

```
ld a, [$200]
add a, [$201]
ld [$203], a
jmp cont
ld [$202], a
```

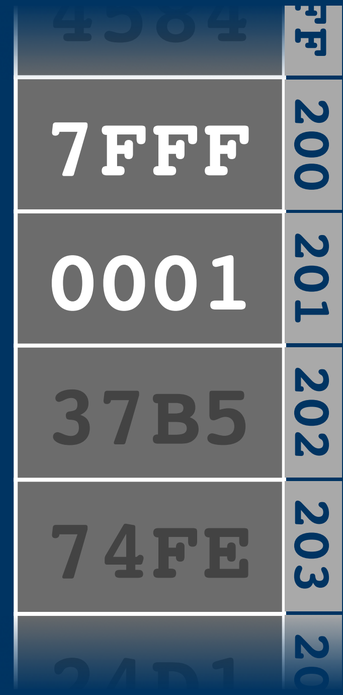


Overflow Handling

A	7FFF
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000



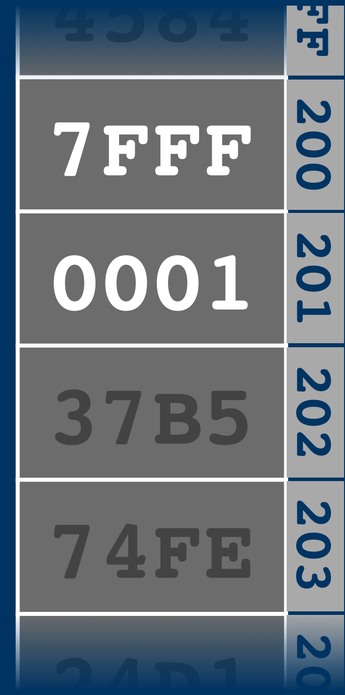

```
    ld a, [$200]
PC→ add a, [$201]
    ld [$203], a
    jmp cont
    ld [$202], a
```



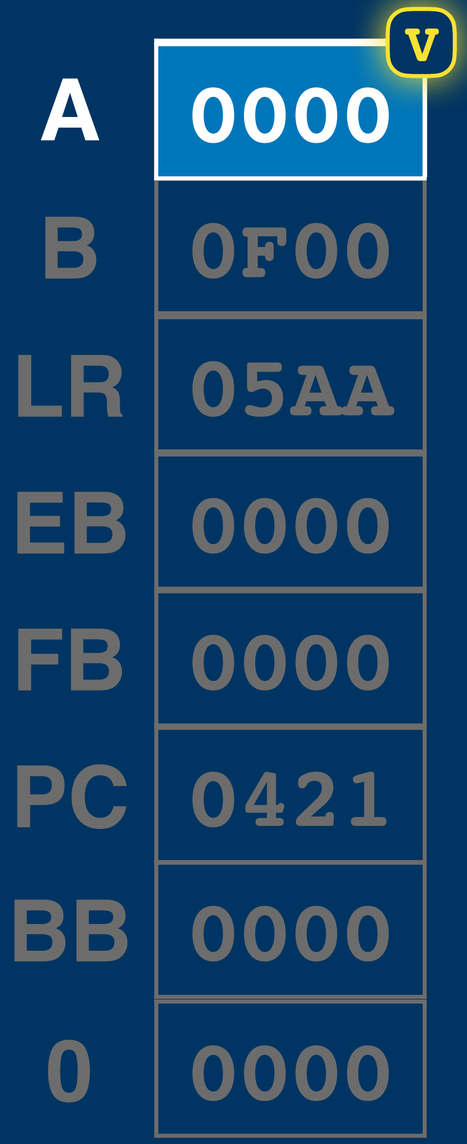
Overflow Handling

A	0000 V
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000


```
    ld a, [$200]
PC→ add a, [$201]
    ld [$203], a
    jmp cont
    ld [$202], a
```

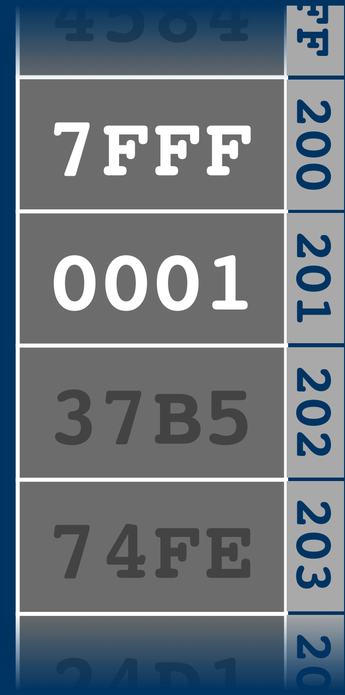


Overflow Handling

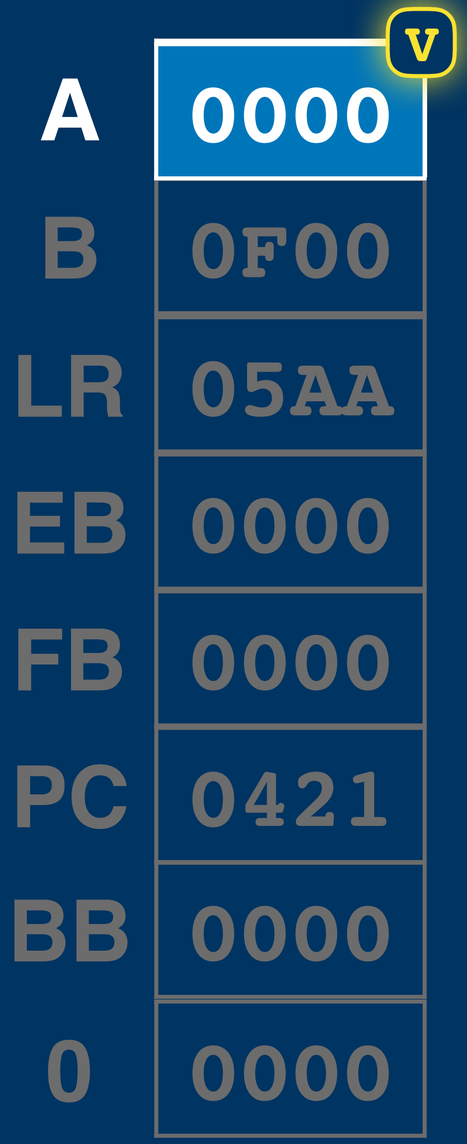



```
ld a, [$200]
add a, [$201]
ld [$203], a
jmp cont
ld [$202], a
```

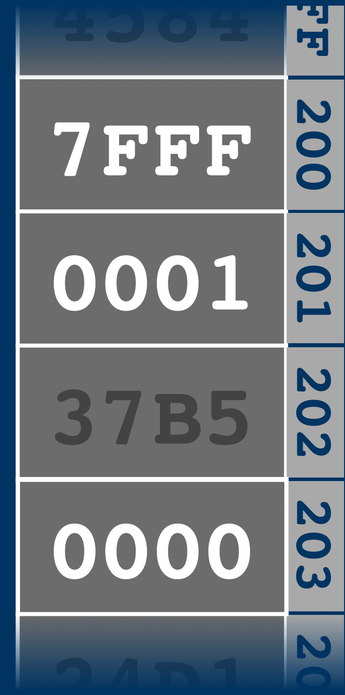
PC →



Overflow Handling



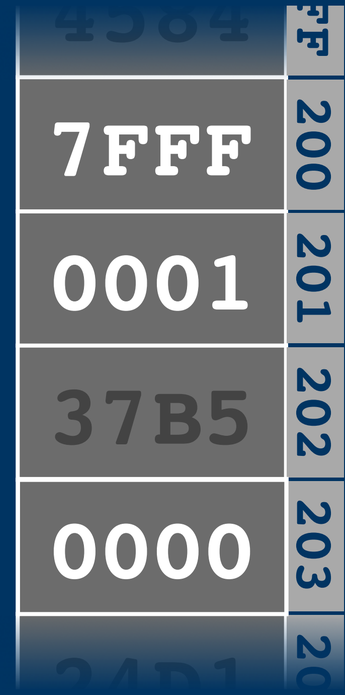

```
ld a, [$200]
add a, [$201]
PC→ ld [$203], a
jmp cont
ld [$202], a
```



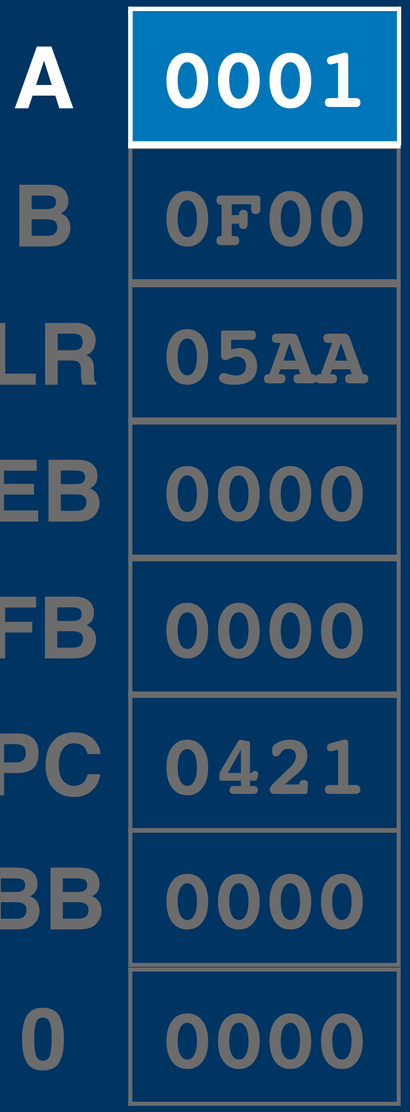
Overflow Handling

A	0000
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

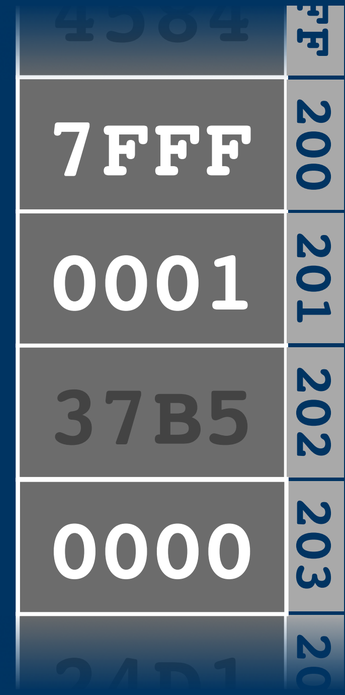

```
ld a, [$200]
add a, [$201]
PC→ ld [$203], a
jmp cont
ld [$202], a
```



Overflow Handling




```
ld a, [$200]
add a, [$201]
PC→ ld [$203], a
jmp cont
ld [$202], a
```



Overflow Handling

A	0001
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000


```
ld a, [$200]
add a, [$201]
ld [$203], a
jmp cont
ld [$202], a
```

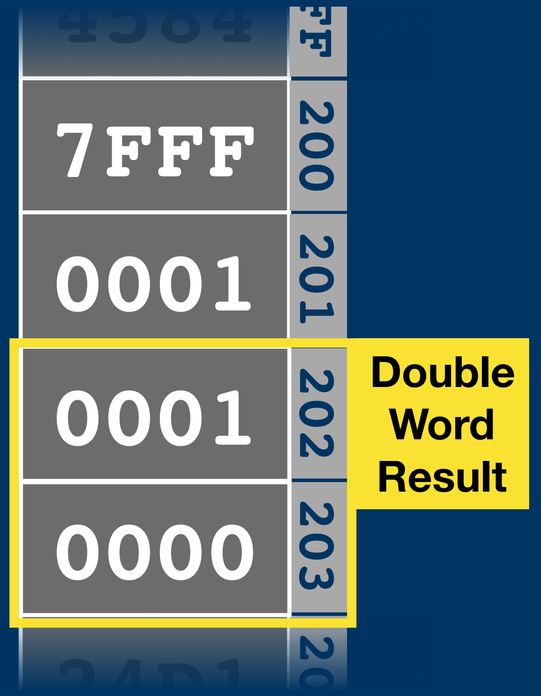
PC →

7FFF	200
0001	201
37B5	202
0000	203

Overflow Handling

A	0001
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000


```
ld a, [$200]
add a, [$201]
ld [$203], a
jmp cont
ld [$202], a
```



Load Complement

```
ldc a, [$200]
```

Load Cpl	ldc a, [k]	CS k
	4000+k	2
	A ← -M[k]	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
Z	0421
BB	0000
0	0000

Load Complement

```
ldc a, [$200]
```

Load Cpl	ldc a, [k]	CS k
	4000+k	2
	A ← −M[k]	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
Z	0421
BB	0000
0	0000

1111

281F	1FC
0CDC	1FD
3193	1FE
4584	1FF
1111	200
0A0A	201
37B5	202
52A3	203
24D1	204

Load Complement

`ldc a, [$200]`

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
Z	0421
BB	0000
0	0000

6EEE

1FC	281F
1FD	0CDC
1FE	3193
1FF	4584
200	1111
201	0A0A
202	37B5
203	52A3
204	24D1

Load Cpl		
	ldc a, [k]	CS k
	4000+k	2
	A ← −M[k]	

Load Complement

`ldc a, [$200]`

A	6EEE
B	0F00
LR	05AA
EB	0000
FB	0000
Z	0421
BB	0000
0	0000

	281F	1FC
	0CDC	1FD
	3193	1FE
	4584	1FF
	1111	200
	0A0A	201
	37B5	202
	52A3	203
	24D1	204

Load Cpl		
	ldc a, [k]	CS k
	4000+k	2
	A ← −M[k]	

Increment

```
inc [$200]
```

Increment	inc [k]	INCR k
	2800+k	2
	M[k] ← M[k]+1	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Increment

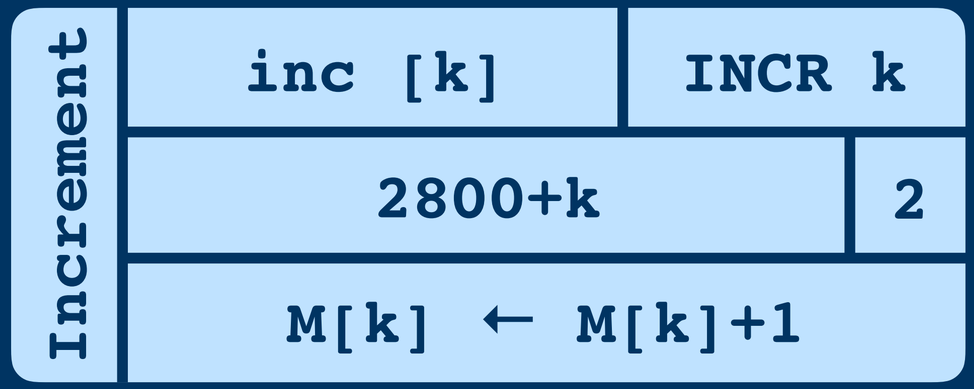
`inc [$200]`

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

281F	1FC	
0CDC	1FD	
3193	1FE	
4584	1FF	
1112	200	
0A0A	201	
37B5	202	
52A3	203	
24D1	204	

Increment		
	inc [k]	INCR k
	2800+k	2
	M[k] ← M[k]+1	

Augment/Diminish



Augment/Diminish

Increment

inc [k]	INCR k
2800+k	2
M[k] ← M[k]+1	

Augment

aug [k]	AUG k
0006, 2800+k	3

```
if M[k]≥+0:
    M[k] ← M[k]+1
if M[j]≤-0:
    M[k] ← M[k]-1
```

Diminish

dim [k]	DIM k
0006, 2C00+k	3

```
if M[k]>+0:
    M[k] ← M[k]-1
if M[j]<-0:
    M[k] ← M[k]+1
```

Inc/Augment/Diminish

Increment

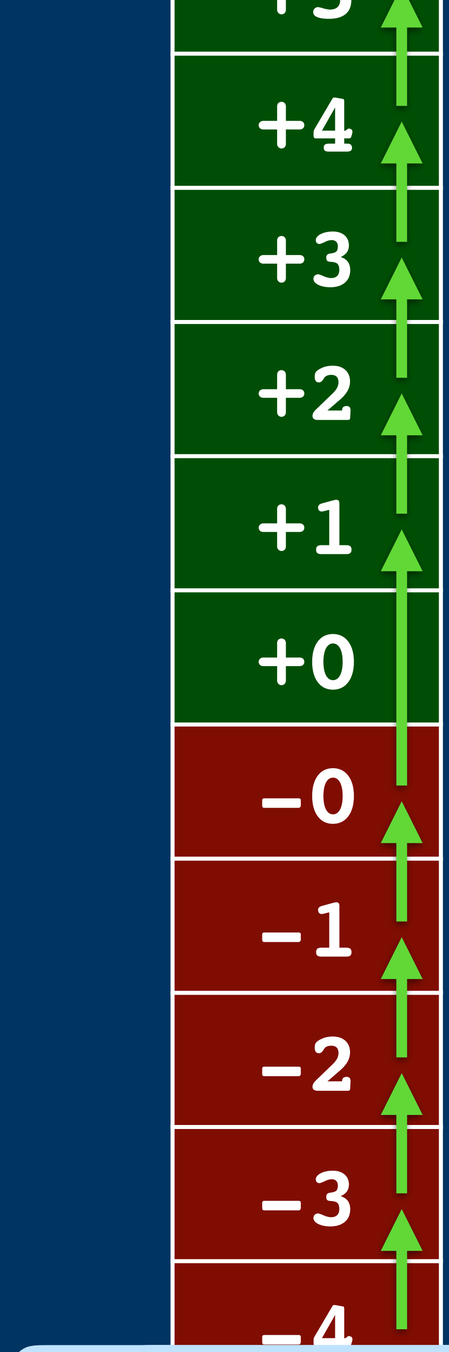
Augment

Diminish

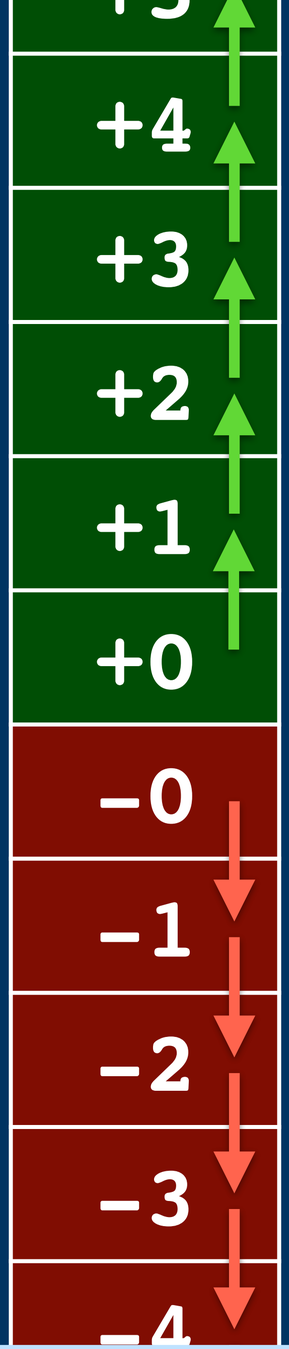
Inc/Augment/Diminish

Inc/Augment/Diminish

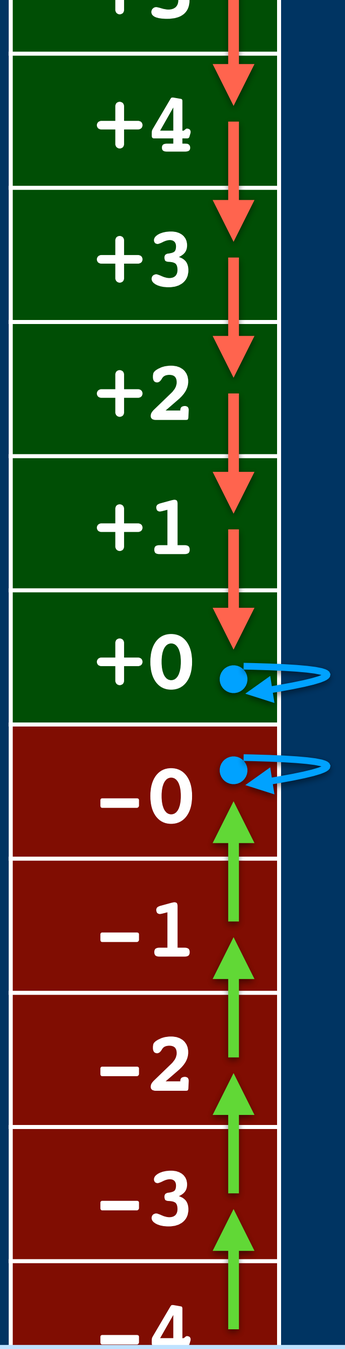
Inc/Augment/Diminish



Increment



Augment



Diminish

Multiply

```
mul [$200]
```

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Multiply		
	mul [k]	MP k
	0006, 7000+k	4
	A,B ← A × M[k]	

Multiply

```
mul [$200]
```

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

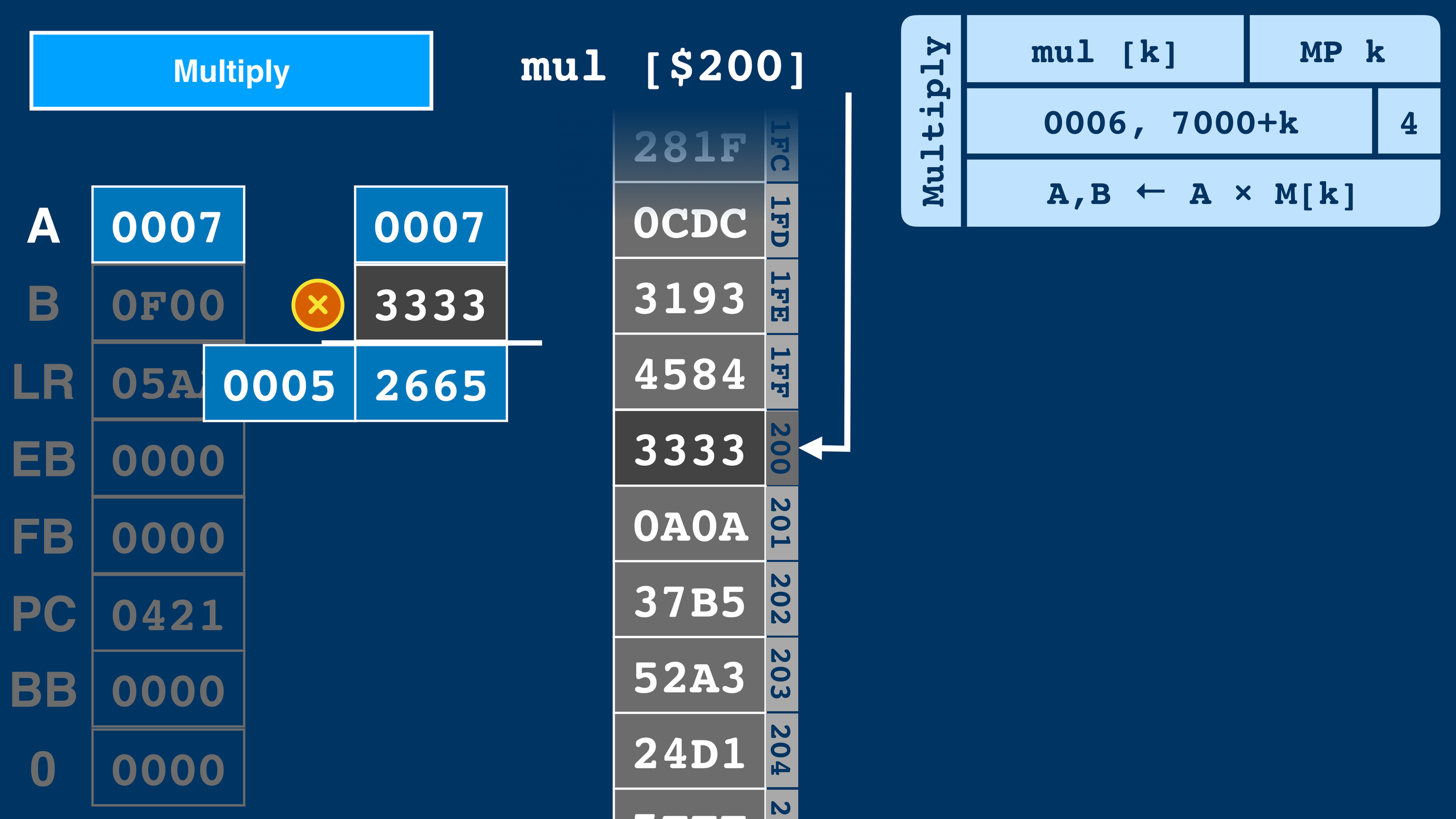
	0007
×	3333
	2665

1FC	281F
1FD	0CDC
1FE	3193
1FF	4584
200	3333
201	0A0A
202	37B5
203	52A3
204	24D1

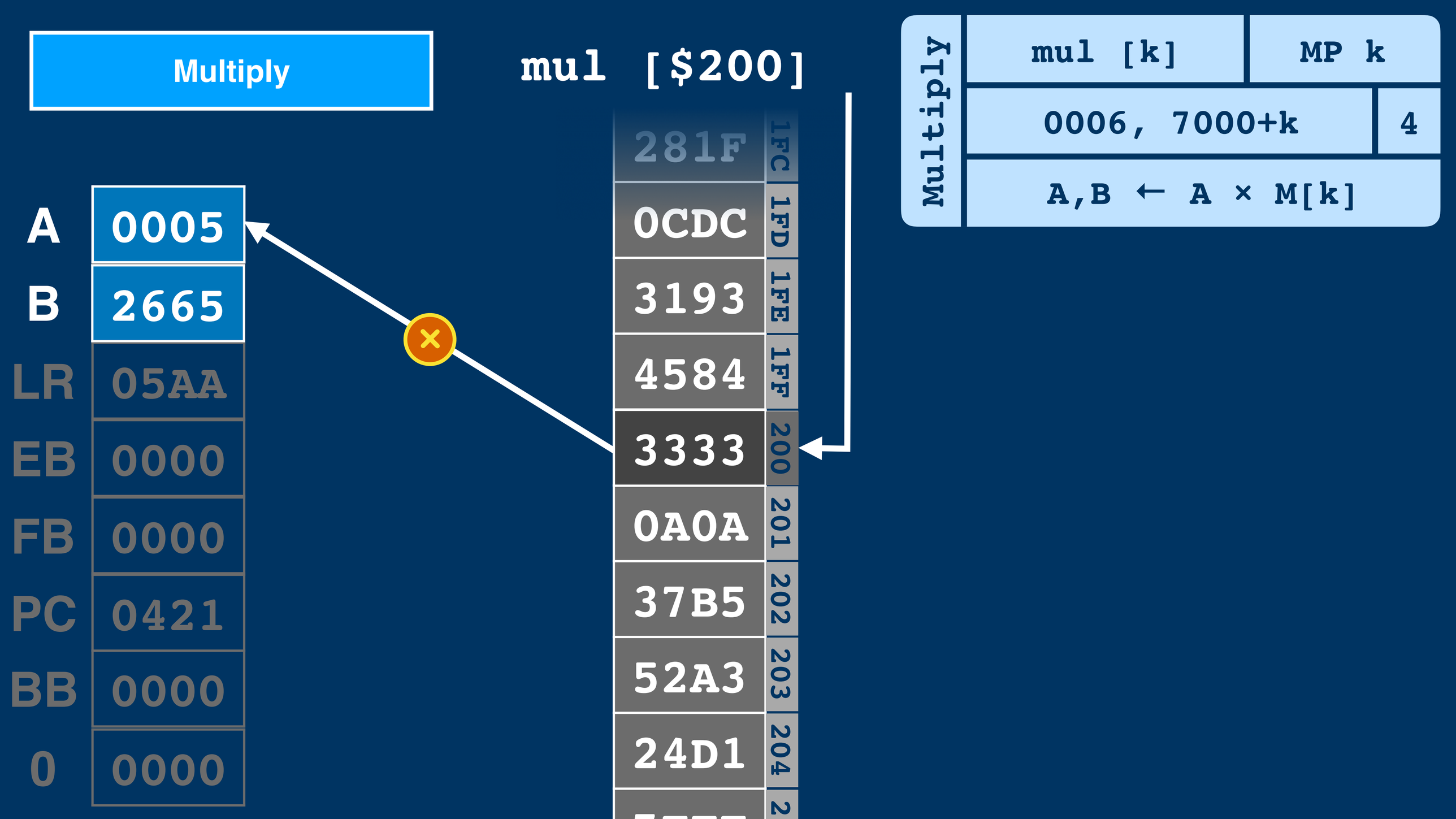
Multiply

mul [k]	MP k
0006, 7000+k	4
A,B ← A × M[k]	

Multiply


mul [$200]
A 0007
B 0F00
LR 05A
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
0007
3333
0005
2665
281F
0CDC
3193
4584
3333
0A0A
37B5
52A3
24D1
1FC
1FD
1FE
1FF
200
201
202
203
204
Multiply
mul [k]
MP k
0006, 7000+k
4
A,B ← A × M[k]

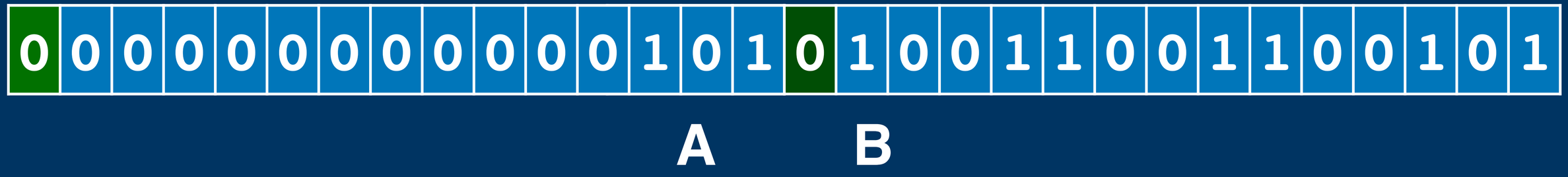
Multiply


mul [$200]
A 0005
B 2665
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
281F 1FC
0CDC 1FD
3193 1FE
4584 1FF
3333 200
0A0A 201
37B5 202
52A3 203
24D1 204
Multiply
mul [k]
MP k
0006, 7000+k
4
A,B ← A × M[k]

Double Word Values

Double Word Values

Double Word Values



Double Word Values

```
($0005 << 14) | $2665
```


```
= $16665
```

Divide

```
div [$200]
```

A	0000
B	0007
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Divide

div [k]	DV k
0006, 1000+k	7

A ← A,B ÷ M[k]
B ← A,B % M[k]

Divide

```
div [$200]
```

A	000(	0000	0007
B	0007	÷	0003
			0002

LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

281F	1FC
0CDC	1FD
3193	1FE
4584	1FF
0003	200
0A0A	201
37B5	202
52A3	203
24D1	204

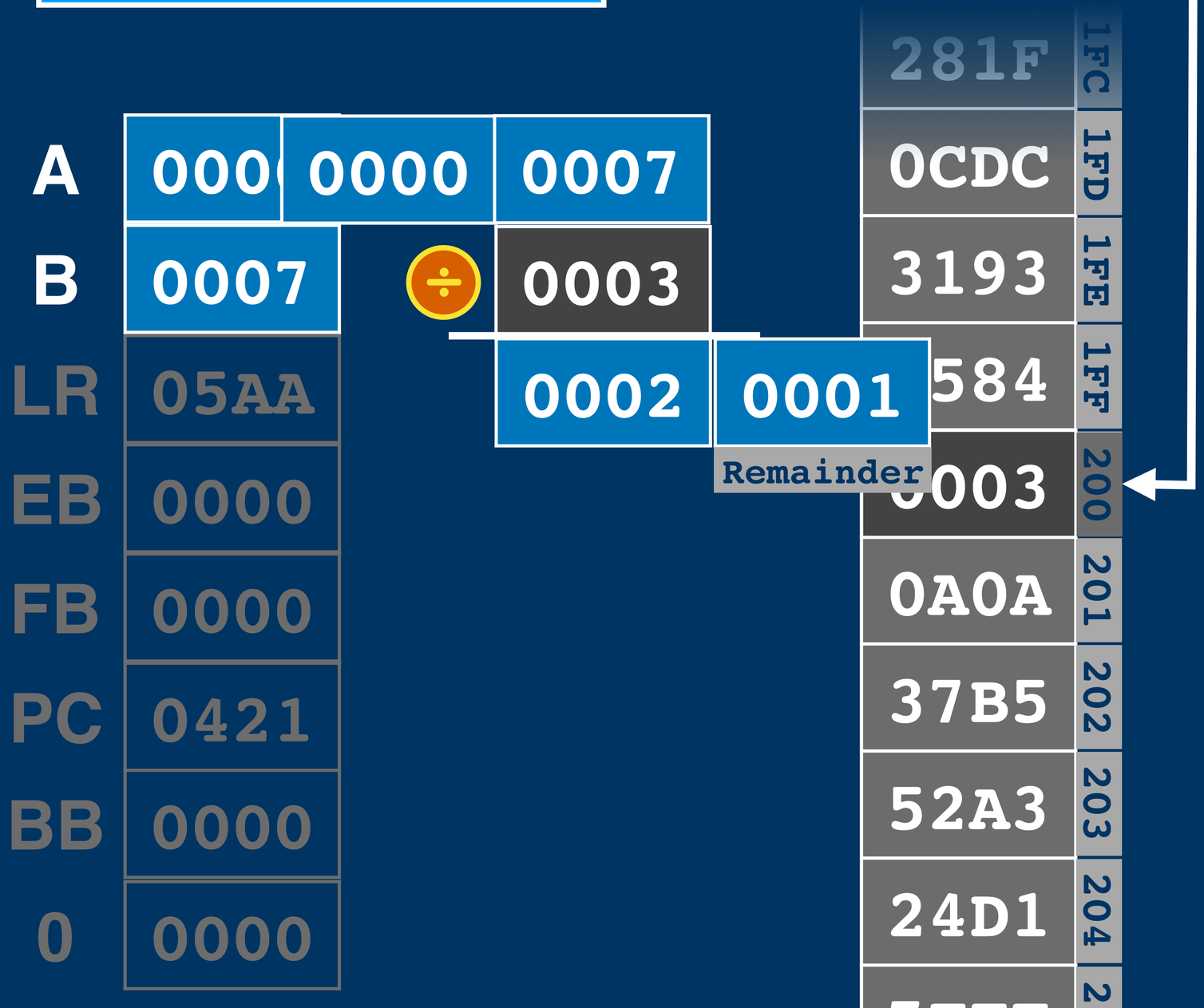
Divide

`div [k]`	`DV k`
`0006, 1000+k`	`7`

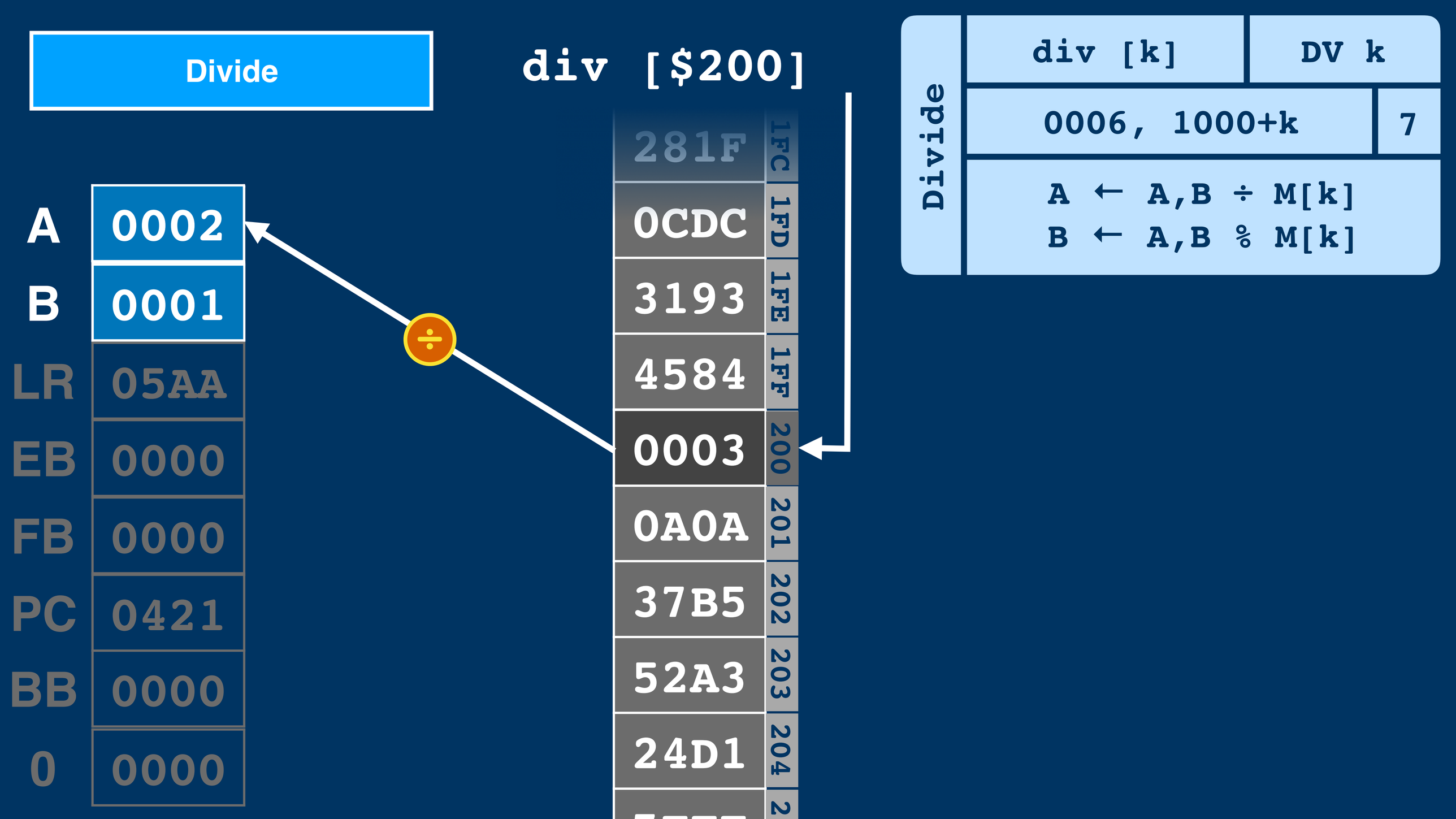
`A ← A,B ÷ M[k]`
`B ← A,B % M[k]`

Divide

div [$200]



Divide	div [k]	DV k
	0006, 1000+k	7
	A ← A,B ÷ M[k] B ← A,B % M[k]	


Divide
div [$200]
A 0002
B 0001
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
÷
281F 1FC
0CDC 1FD
3193 1FE
4584 1FF
0003 200
0A0A 201
37B5 202
52A3 203
24D1 204
Divide
div [k]
DV k
0006, 1000+k
7
A ← A,B ÷ M[k]
B ← A,B % M[k]

Load Double

```
ld ab, [$200]
```

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Load Double

ld ab, [k]	DCA k
0006, 3001+k	4
A ← M[k] B ← M[k+1]	

Load Double

```
ld ab, [$200]
```

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Load Double		
	ld ab, [k]	DCA k
	0006, 3001+k	4
	A ← M[k] B ← M[k+1]	

Load Double

```
ld ab, [$200]
```

A	1111
B	0A0A
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

1FC	281F
1FD	0CDC
1FE	3193
1FF	4584
200	1111
201	0A0A
202	37B5
203	52A3
204	24D1

Load Double

ld ab, [k]	DCA k
0006, 3001+k	4

A ← M[k]
B ← M[k+1]

Load Double

```
ld ab, [$200]
```

A	1111
B	0A0A
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

281F	1FC	
0CDC	1FD	
3193	1FE	
4584	1FF	
1111	200	←
0A0A	201	←
37B5	202	
52A3	203	
24D1	204	

Load Double

ld ab, [k]	DCA k	
0006, 3001+k		4
A ← M[k] B ← M[k+1]		

Ld Dbl Cpl

ldc ab, [k]	DCS k	
0006, 4001+k		4
A ← -M[k] B ← -M[k+1]		

Exchange Double

```
xchg ab, [$200]
```

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Xchg Double

xchg ab, [k]	DXCH k
5201+k	3
A ↔ M[k] B ↔ M[k+1]	

Exchange Double

`xchg ab, [$200]`

A	1111
B	0A0A
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

1FC	281F
1FD	0CDC
1FE	3193
1FF	4584
200	0007
201	0F00
202	37B5
203	52A3
204	24D1

xchg Double

`xchg ab, [k]`	`DXCH k`
`5201+k`	`3`
`A ↔ M[k]` `B ↔ M[k+1]`	

Exchange Double

`xchg ab, [$200]`

A	1111
B	0A0A
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

281F	1FC	
0CDC	1FD	
3193	1FE	
4584	1FF	
0007	200	←
0F00	201	←
37B5	202	
52A3	203	
24D1	204	

Xchg Double

`xchg ab, [k]`	`DXCH k`
5201+k	3
A ↔ M[k] B ↔ M[k+1]	

Add Double

`add [k], ab`	`DAS k`
2001+k	3
M[k,k+1] ← M[k,k+1] + A,B	

Exchange B

```
xchg b, [$200]
```

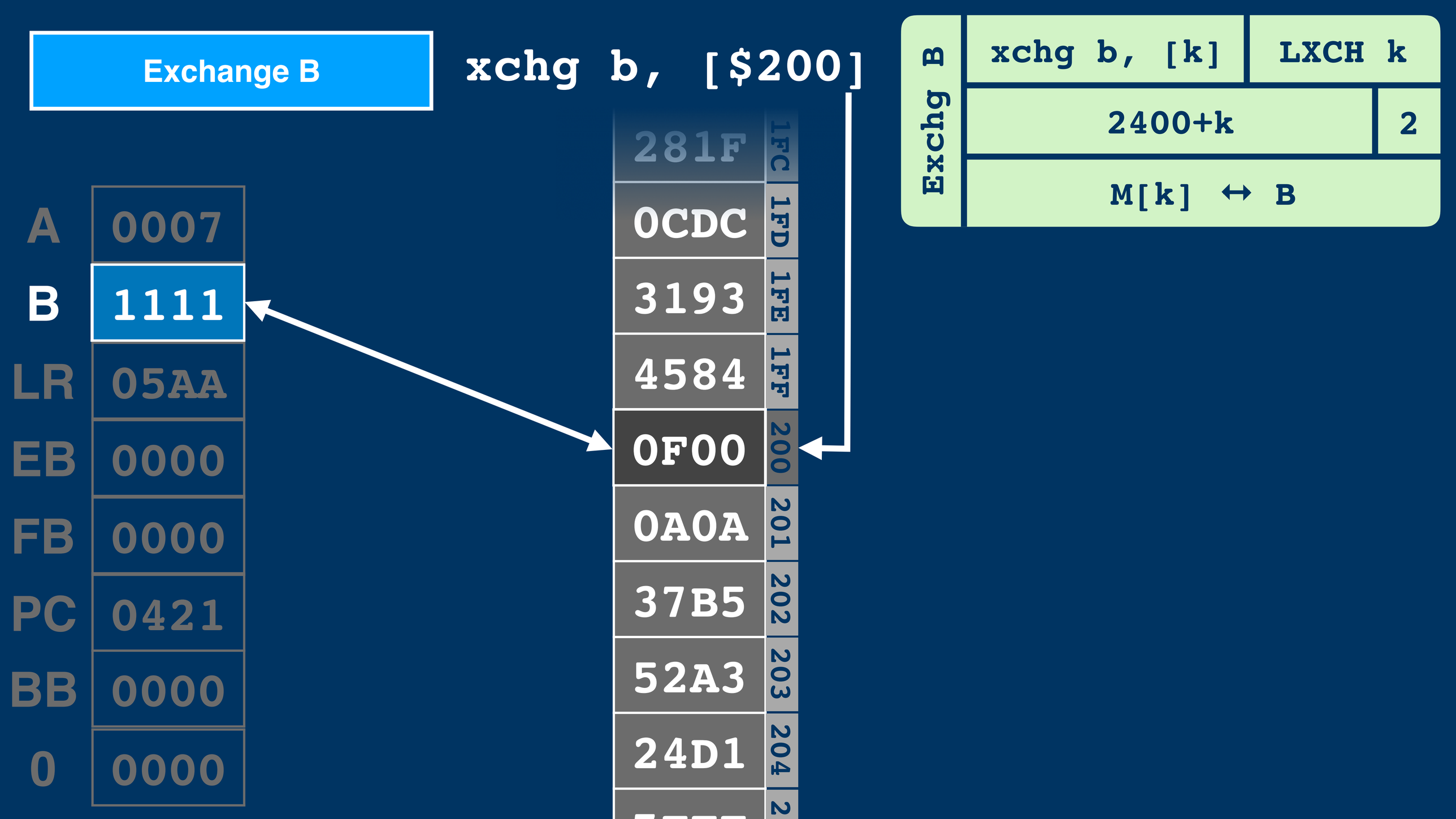
Exchg B	xchg b, [k]	LXCH k
	2400+k	2
	M[k] ↔ B	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Exchange B

```
xchg b, [$200]
```

Exchg B	xchg b, [k]	LXCH k
	2400+k	2
	M[k] ↔ B	



Indexing

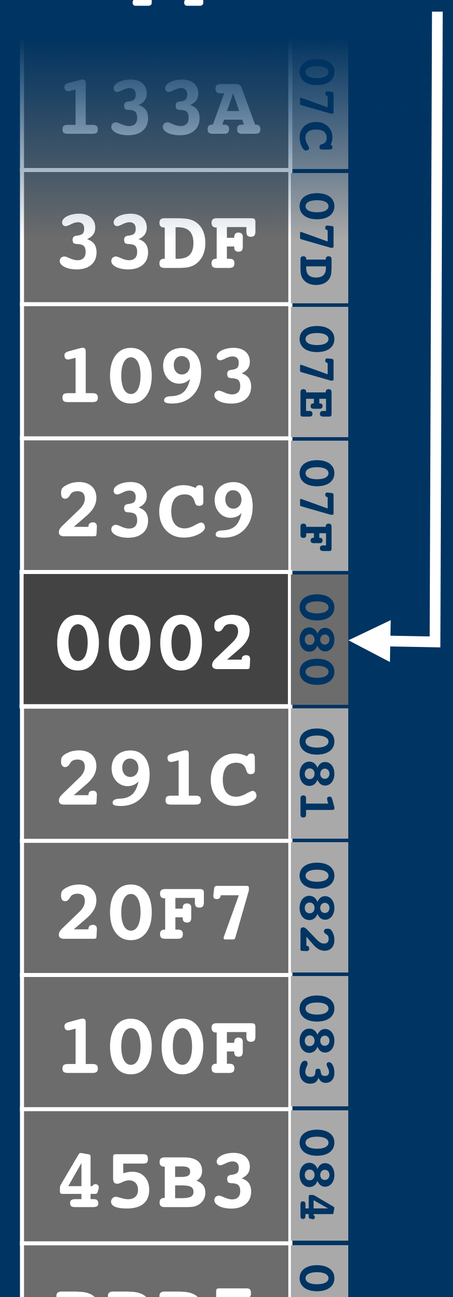
```
ld a, [$700+[$80]]
```

A	0000
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Indexing

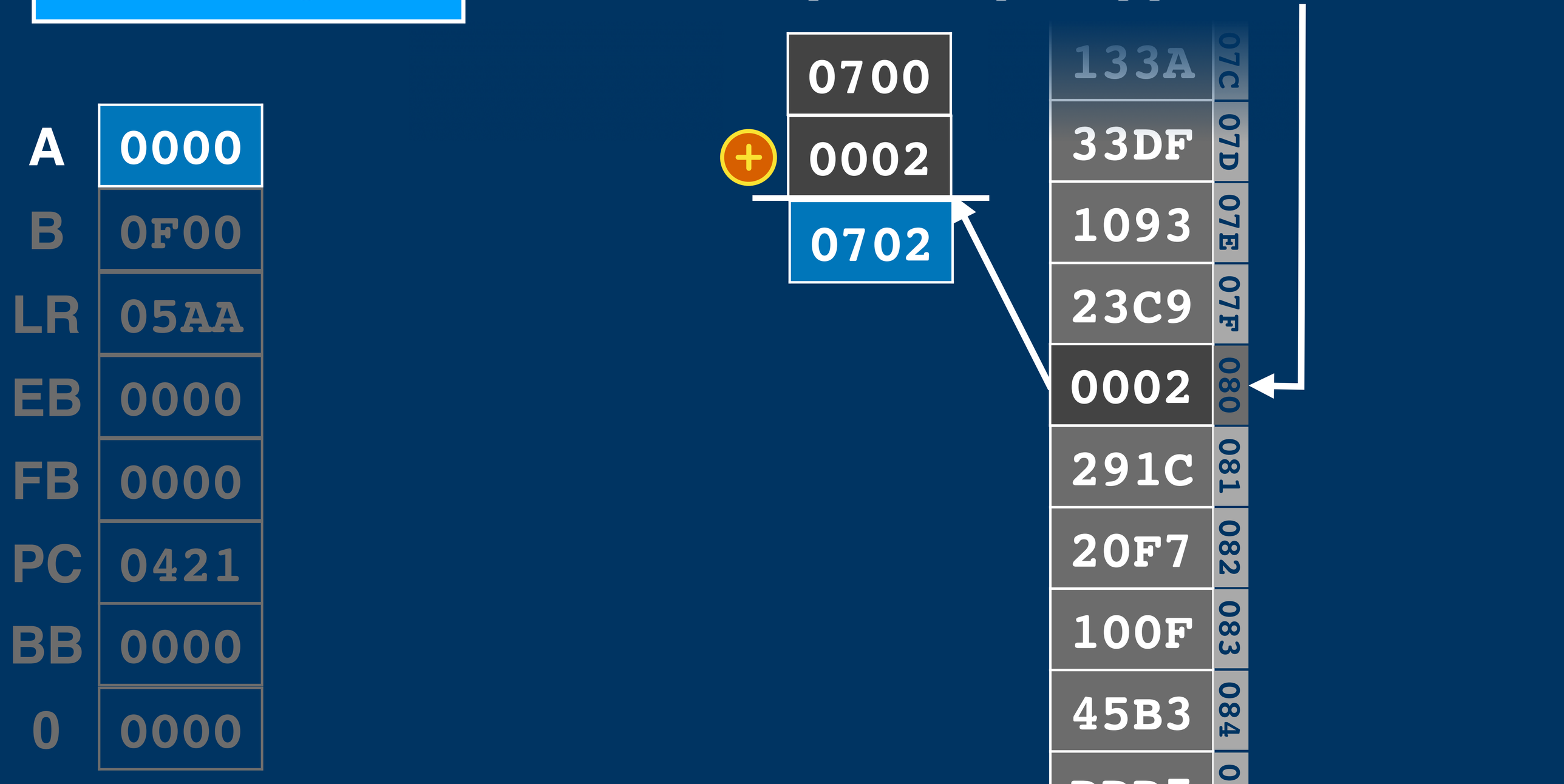
```
ld a, [$700+[$80]]
```

A	0000
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000



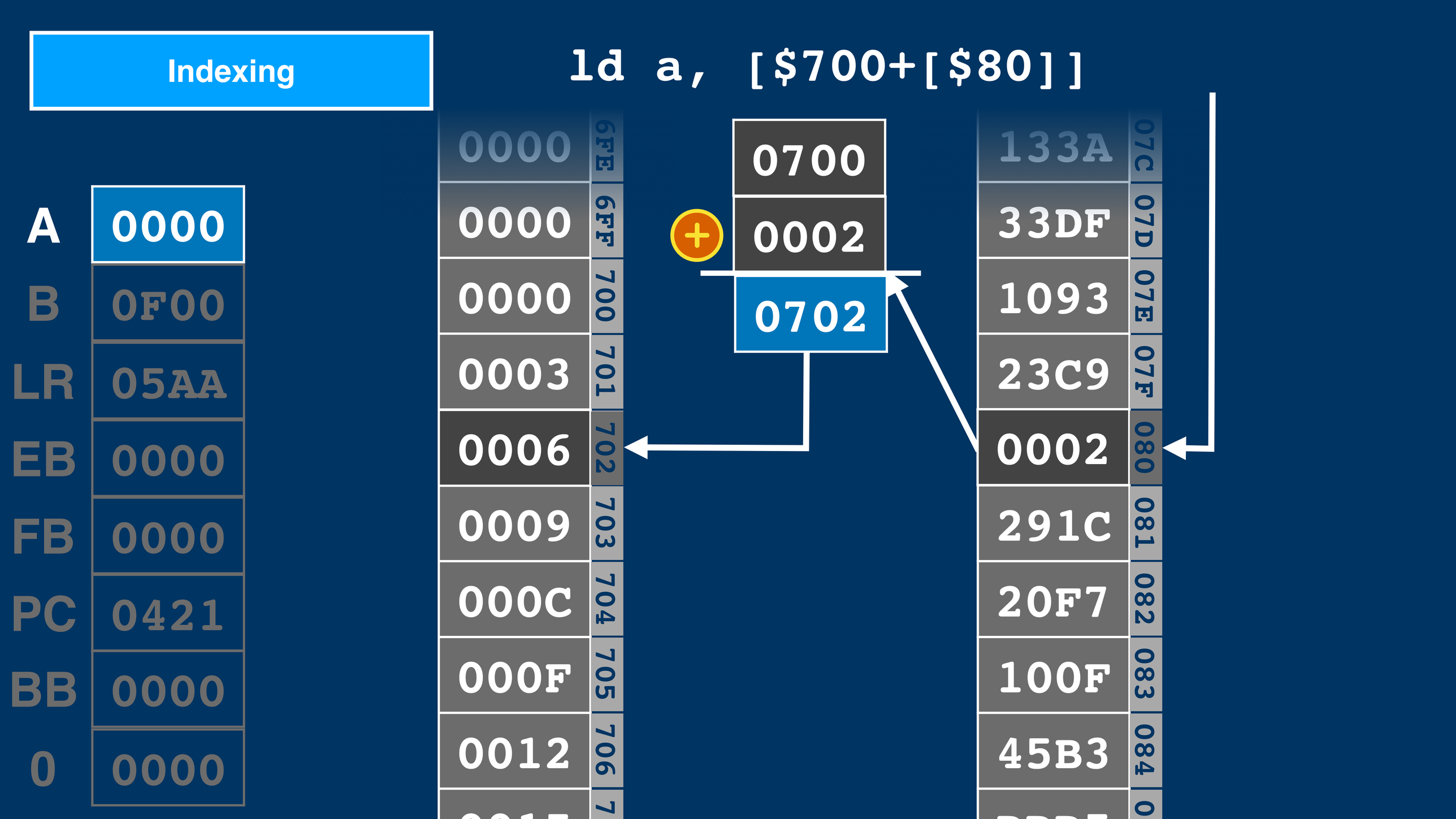
Indexing

```
ld a, [$700+[$80]]
```



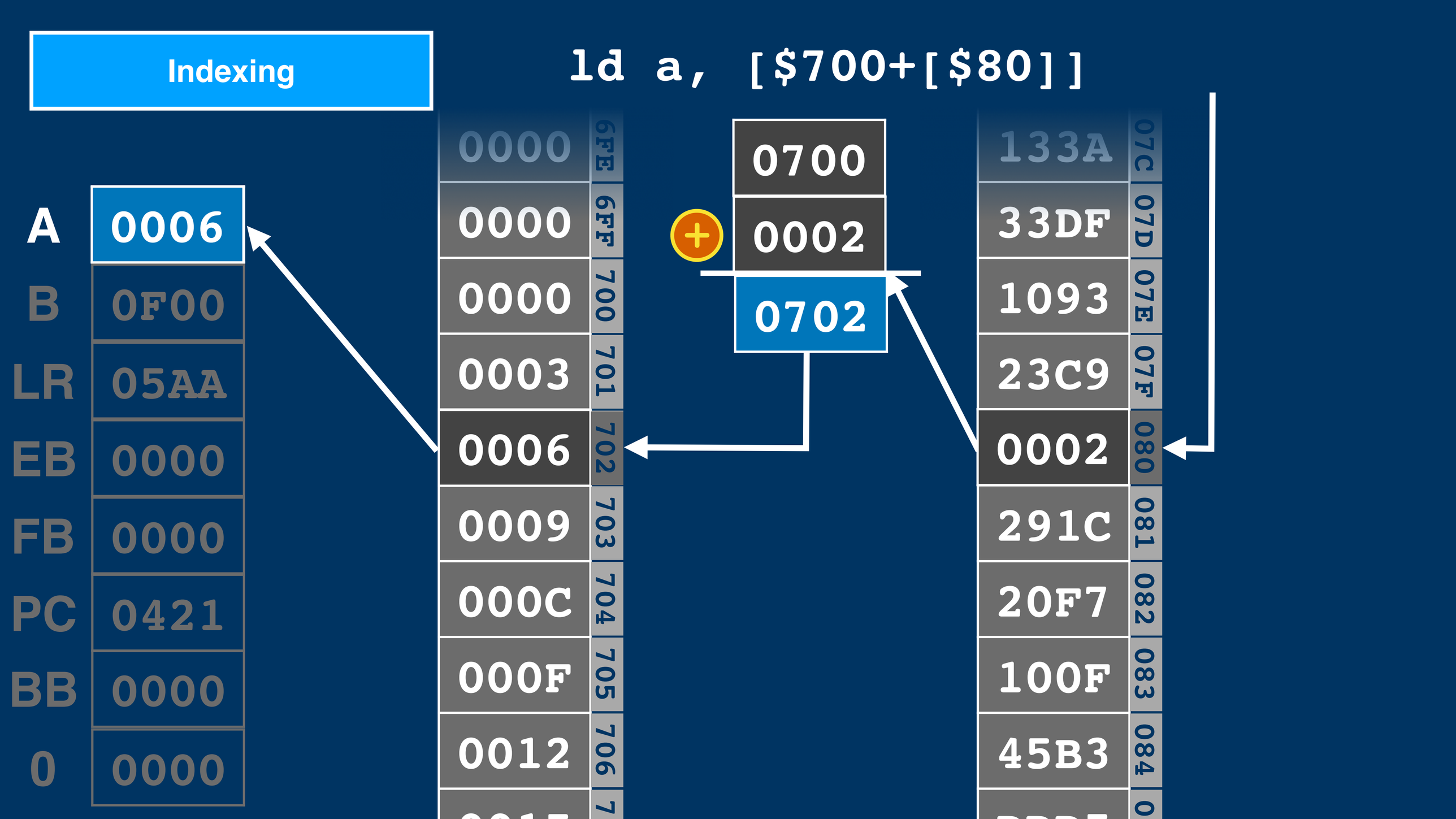
Indexing

```
ld a, [$700+[$80]]
```



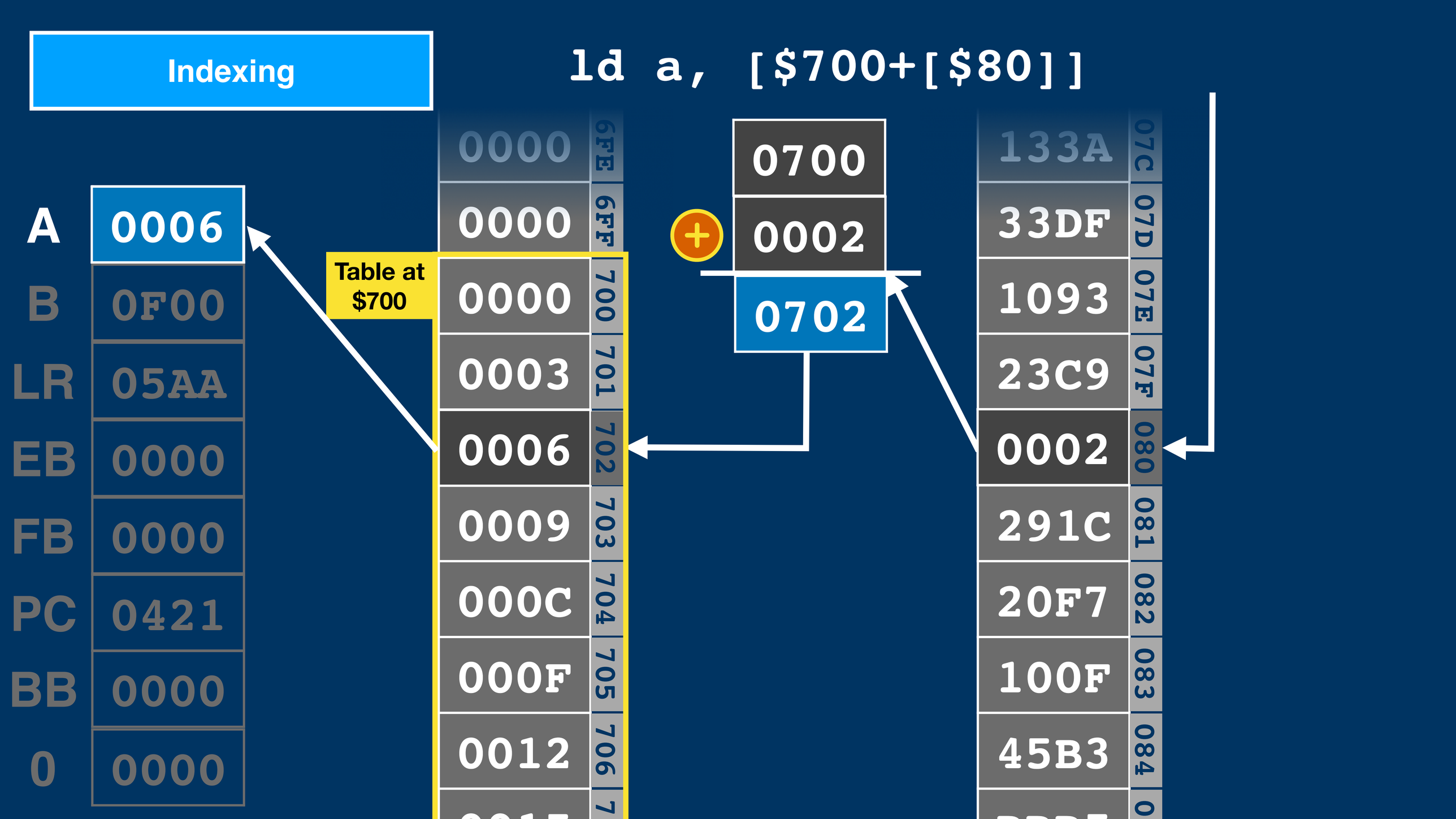
Indexing

```
ld a, [$700+[$80]]
```



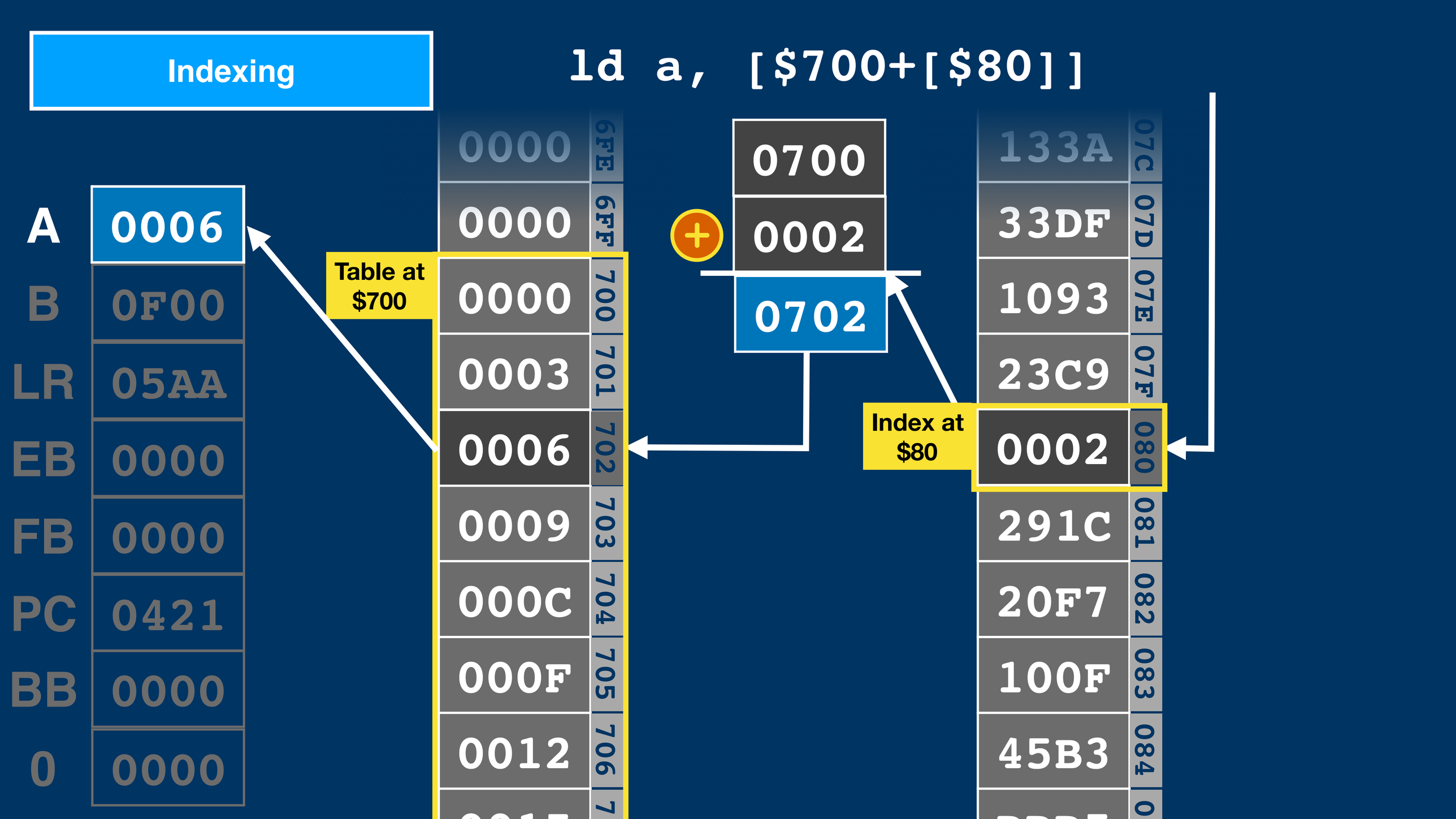
Indexing

```
ld a, [$700+[$80]]
```



Indexing

```
ld a, [$700+[$80]]
```



Indexing

```
ld a, [[$80]]
```

A	0000
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Indexing

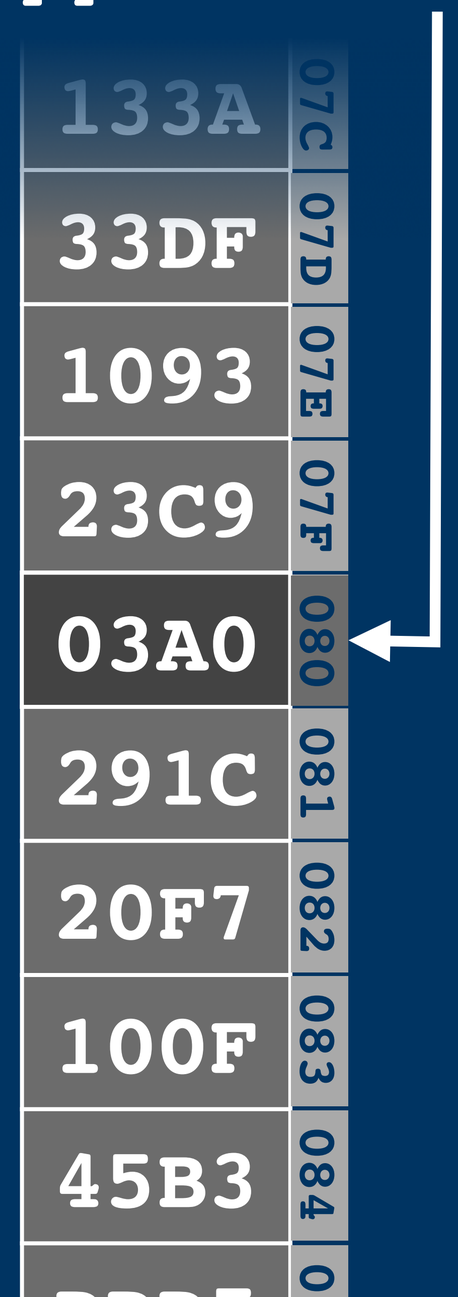
```
ld a, [0+[$80]]
```

A	0000
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Indexing

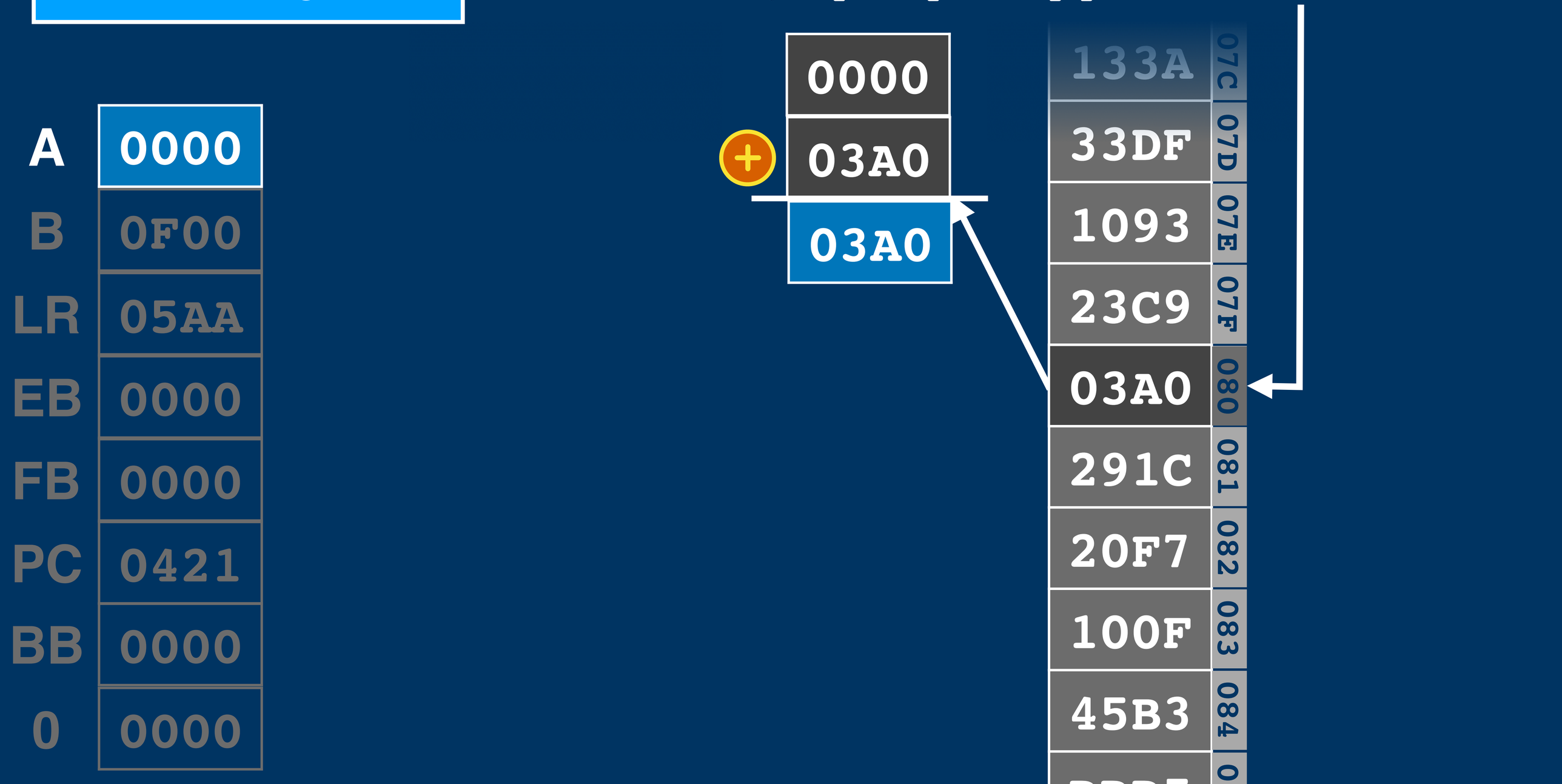
```
ld a, [0+[$80]]
```

A	0000
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000



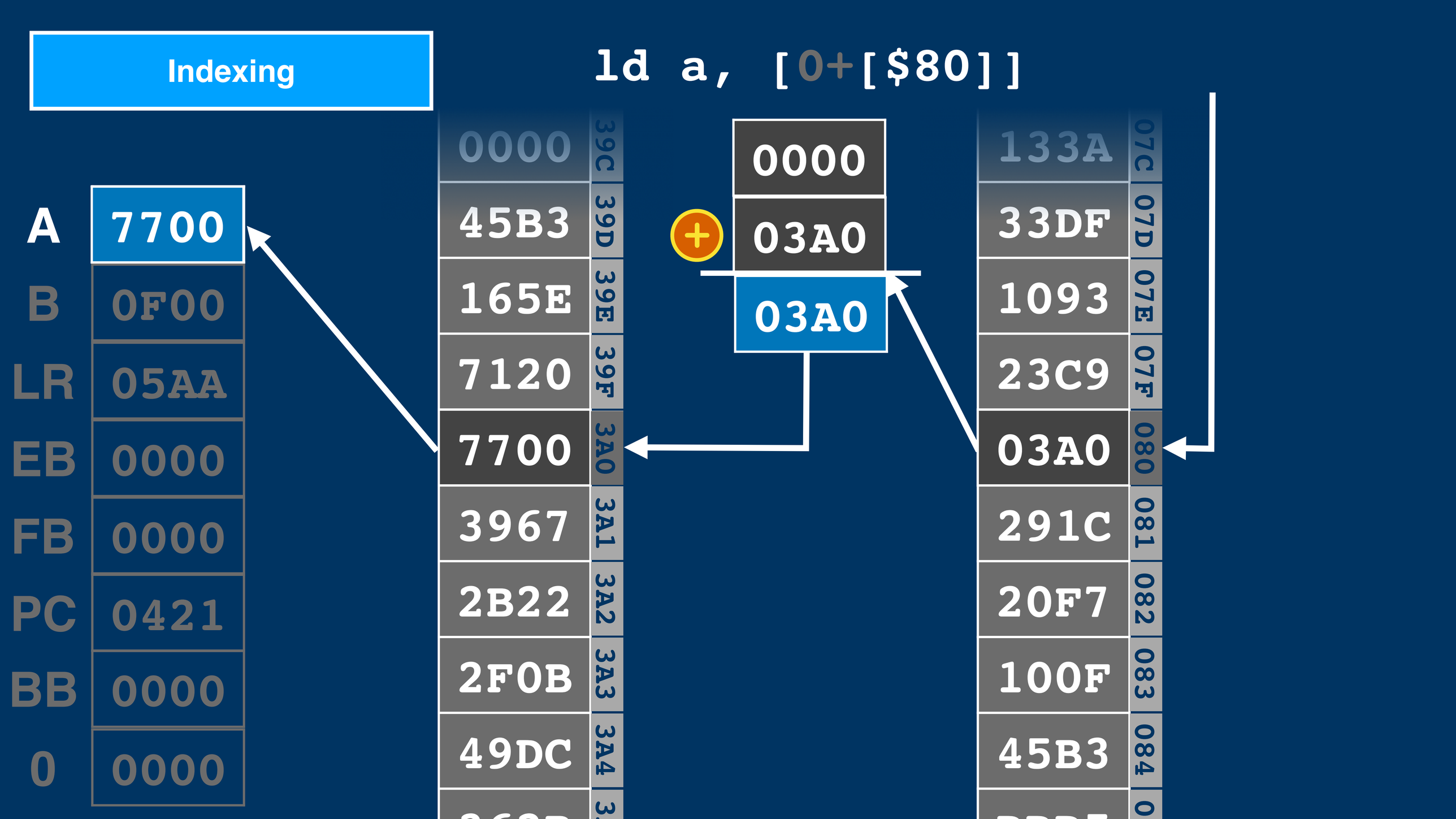
Indexing

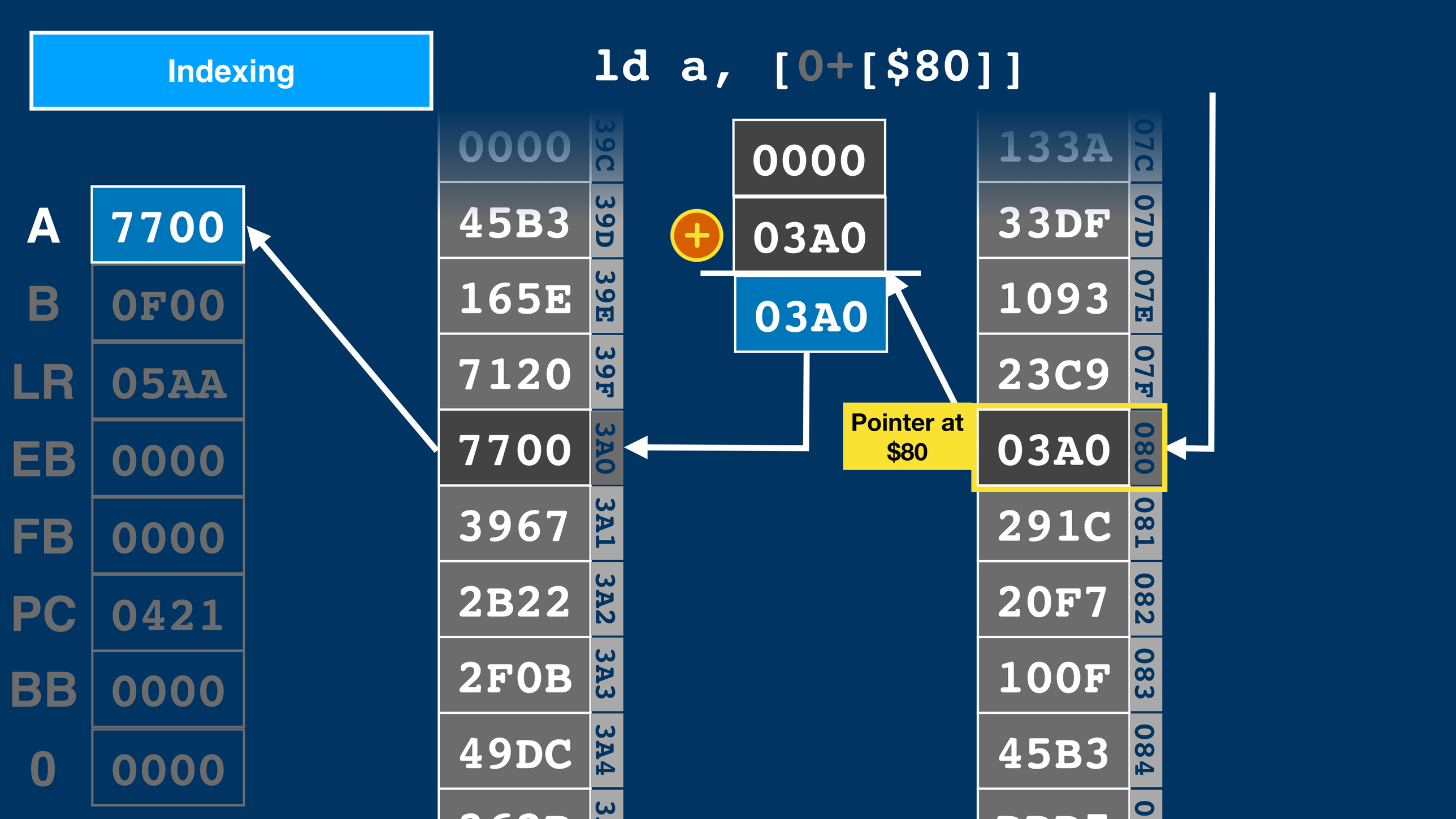
```
ld a, [0+[$80]]
```



Indexing

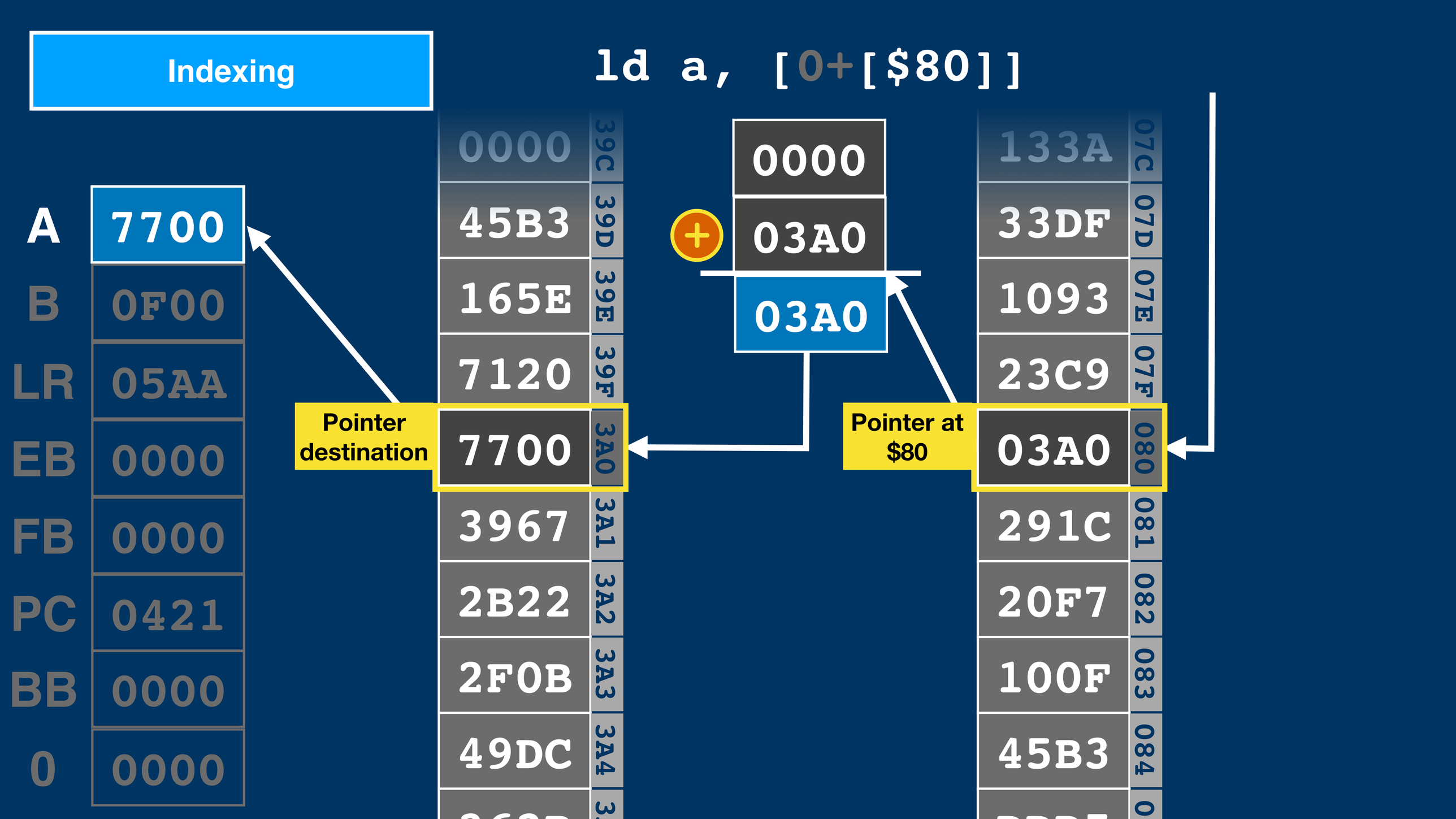
```
ld a, [0+[$80]]
```




Indexing
ld a, [0+[$80]]
A 7700
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
0000 39C
45B3 39D
165E 39E
7120 39F
7700 3A0
3967 3A1
2B22 3A2
2F0B 3A3
49DC 3A4
0000
03A0
03A0
Pointer at $80
133A 07C
33DF 07D
1093 07E
23C9 07F
03A0 080
291C 081
20F7 082
100F 083
45B3 084

Indexing

```
ld a, [0+[$80]]
```



Control Flow

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0400
BB	0000
0	0000

Control Flow

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0400
BB	0000
0	0000

PC →

3100	400	ld a, [$100]
6101	401	add a, [$101]
1405	402	jmp $405
3103	403	ld a, [$103]
6104	404	add a, [$104]
5902	405	ld [$102], a
0006	406	jz $409
140A	407	
3103	408	ld a, [$103]
0006	409	jlez $40f
640F	40A	

Control Flow

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0402
BB	0000
0	0000

PC →

3100	400	ld a, [$100]
6101	401	add a, [$101]
1405	402	jmp $405
3103	403	ld a, [$103]
6104	404	add a, [$104]
5902	405	ld [$102], a
0006	406	jz $409
140A	407	
3103	408	ld a, [$103]
0006	409	jlez $40f
640F	40A	

Control Flow

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0409
BB	0000
0	0000

3100	400	ld a, [$100]
6101	401	add a, [$101]
1405	402	jmp $405
3103	403	ld a, [$103]
6104	404	add a, [$104]
5902	405	ld [$102], a
0006	406	jz $409
140A	407	
3103	408	ld a, [$103]
PC → 0006	409	jlez $40f
640F	40A	

Jump

```
jmp $62f
```

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	041F
BB	0000
0	0000

Jump		
	`jmp k`	`TCF k`
	`1000+k`	`1`
	`PC ← k`	

Jump

jmp $62f

Jump		
	jmp k	TCF k
	1000+k	1
	PC ← k	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	041F
BB	0000
0	0000

PC →	3100	41F	ld a, [$100]
	162F	420	jmp $62f
	3103	421	ld a, [$103]
	5880	62F	ld [$80], a
	3102	630	ld a, [$102]

Jump

jmp $62f

Jump	jmp k	TCF k
	1000+k	1
	PC ← k	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0420
BB	0000
0	0000

	3100	41F	ld a, [$100]
PC →	162F	420	jmp $62f
	3103	421	ld a, [$103]

	5880	62F	ld [$80], a
	3102	630	ld a, [$102]

Jump

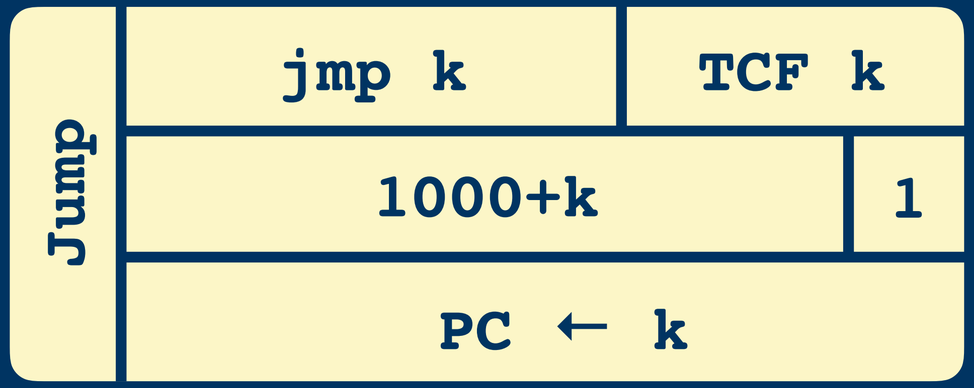
jmp $62f

Jump	jmp k	TCF k
	1000+k	1
	PC ← k	

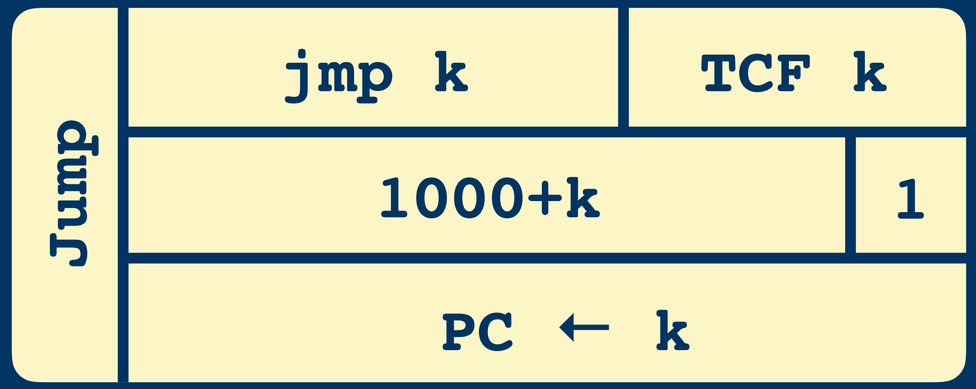
A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	062F
BB	0000
0	0000

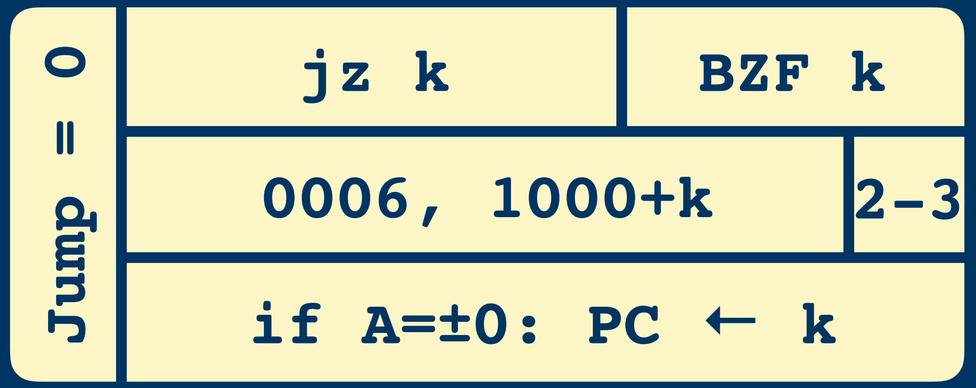
3100	41F	ld a, [$100]
162F	420	jmp $62f
3103	421	ld a, [$103]
PC → 5880	62F	ld [$80], a
3102	630	ld a, [$102]

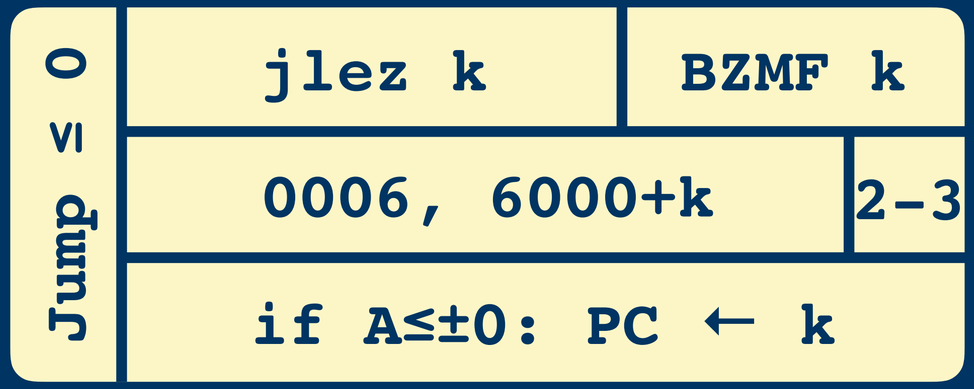
Conditional Jumps



Conditional Jumps







Conditional Jumps

Jump	jmp k	TCF k	
	1000+k	1	
	PC ← k		
Jump = 0	jz k	BZF k	
	0006, 1000+k	2-3	
	if A=±0: PC ← k		
Jump ≤ 0	jlez k	BZMF k	
	0006, 6000+k	2-3	
	if A≤±0: PC ← k		

Conditional Jumps

Jump

jmp k	TCF k
1000+k	1
PC ← k	

Jump = 0

jz k	BZF k
0006, 1000+k	2-3
if A=±0: PC ← k	

Jump ≤ 0

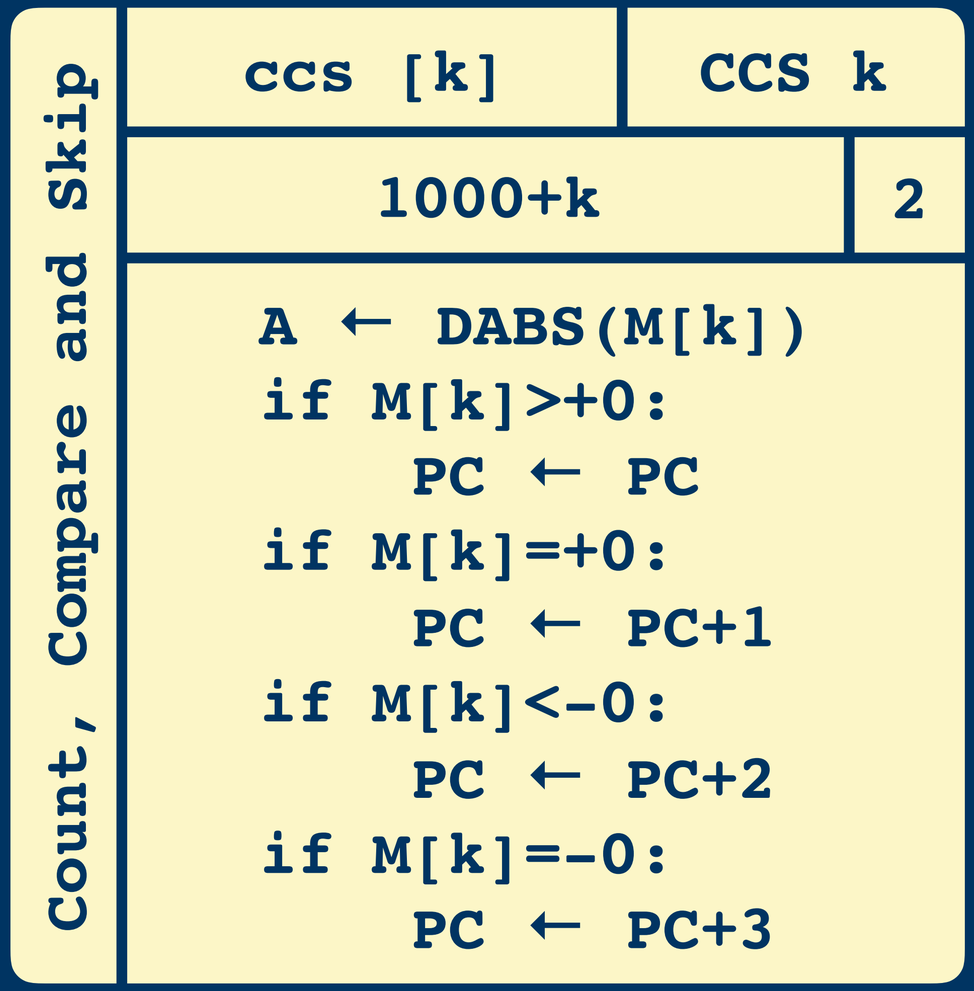
jlez k	BZMF k
0006, 6000+k	2-3
if A≤±0: PC ← k	

Count, Compare and Skip

ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
    PC ← PC
if M[k]=+0:
    PC ← PC+1
if M[k]<-0:
    PC ← PC+2
if M[k]=-0:
    PC ← PC+3
```

Conditional Jumps

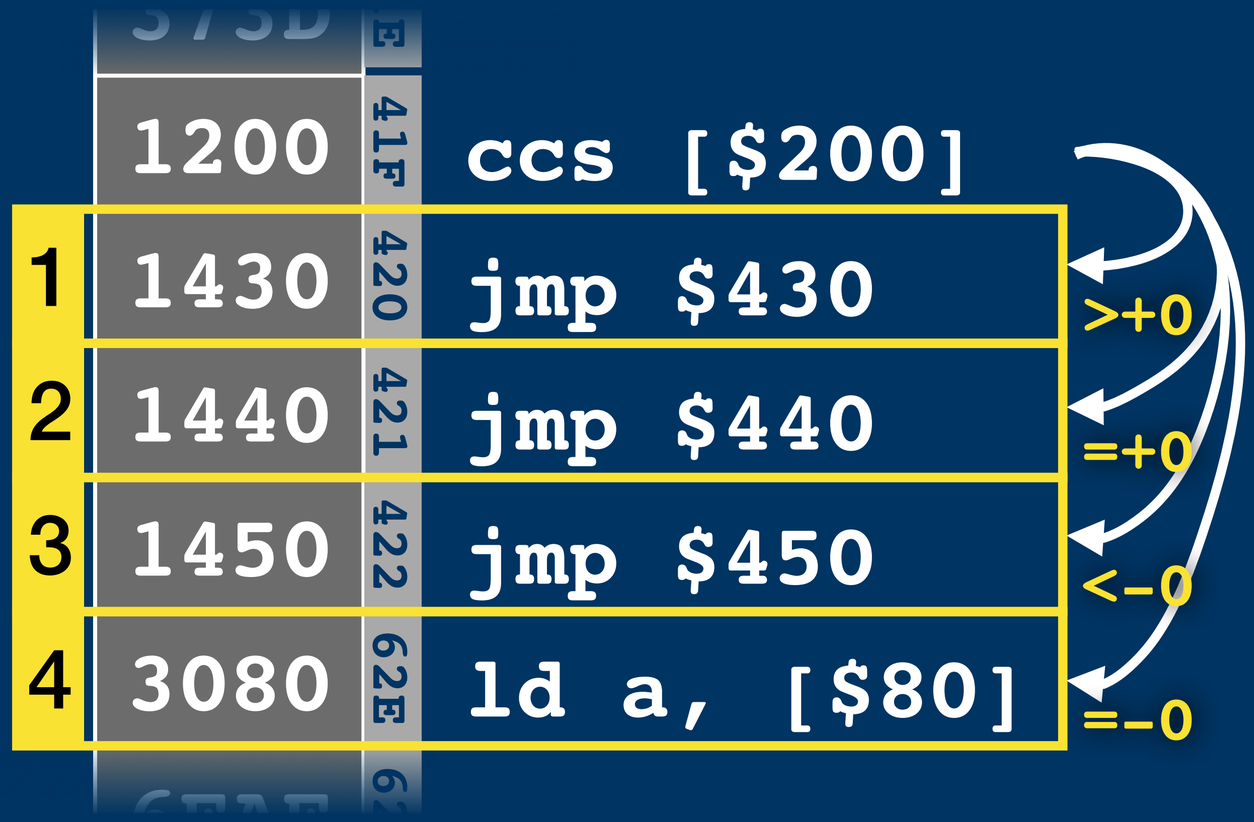


Conditional Jumps

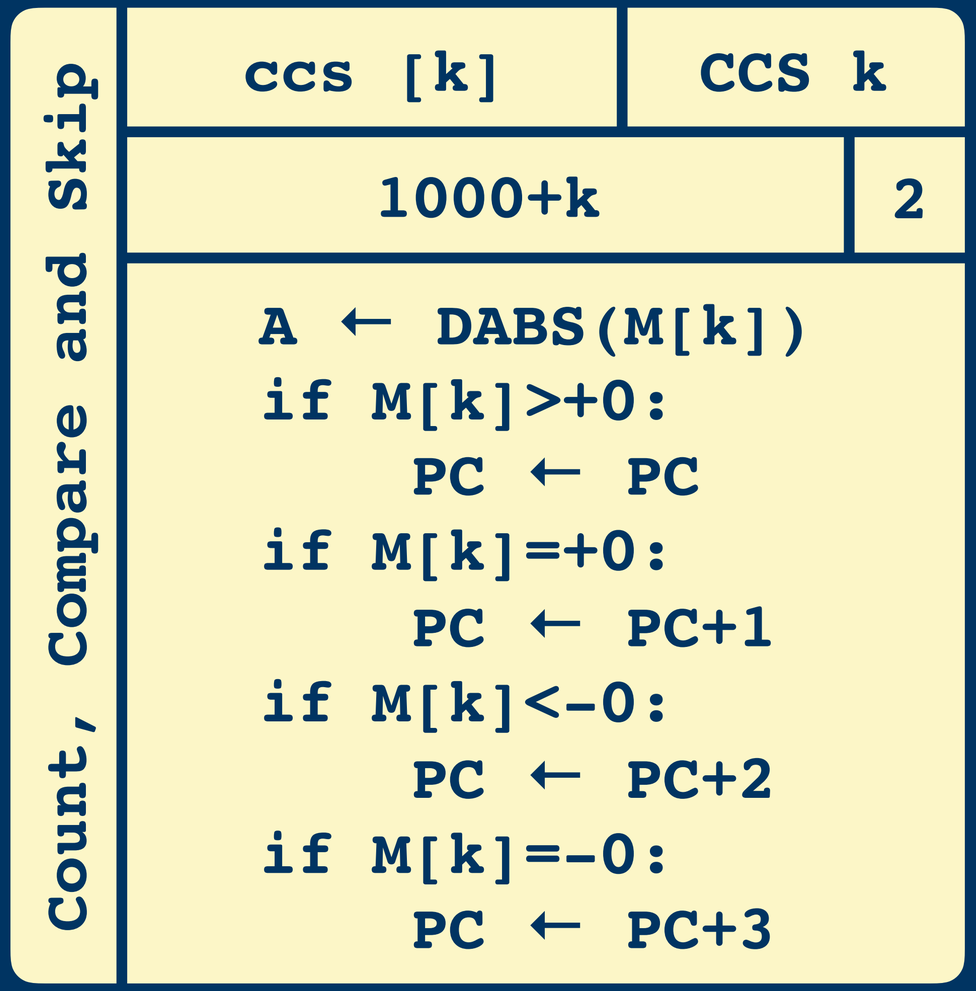
Count, Compare and Skip

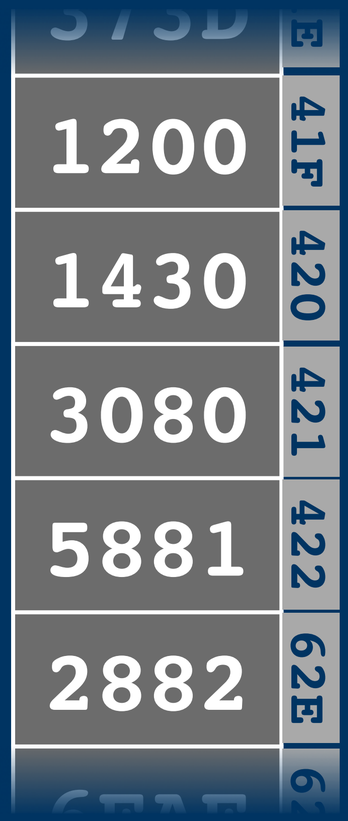
ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
     PC ← PC
if M[k]=+0:
     PC ← PC+1
if M[k]<-0:
     PC ← PC+2
if M[k]=-0:
     PC ← PC+3
```



Conditional Jumps





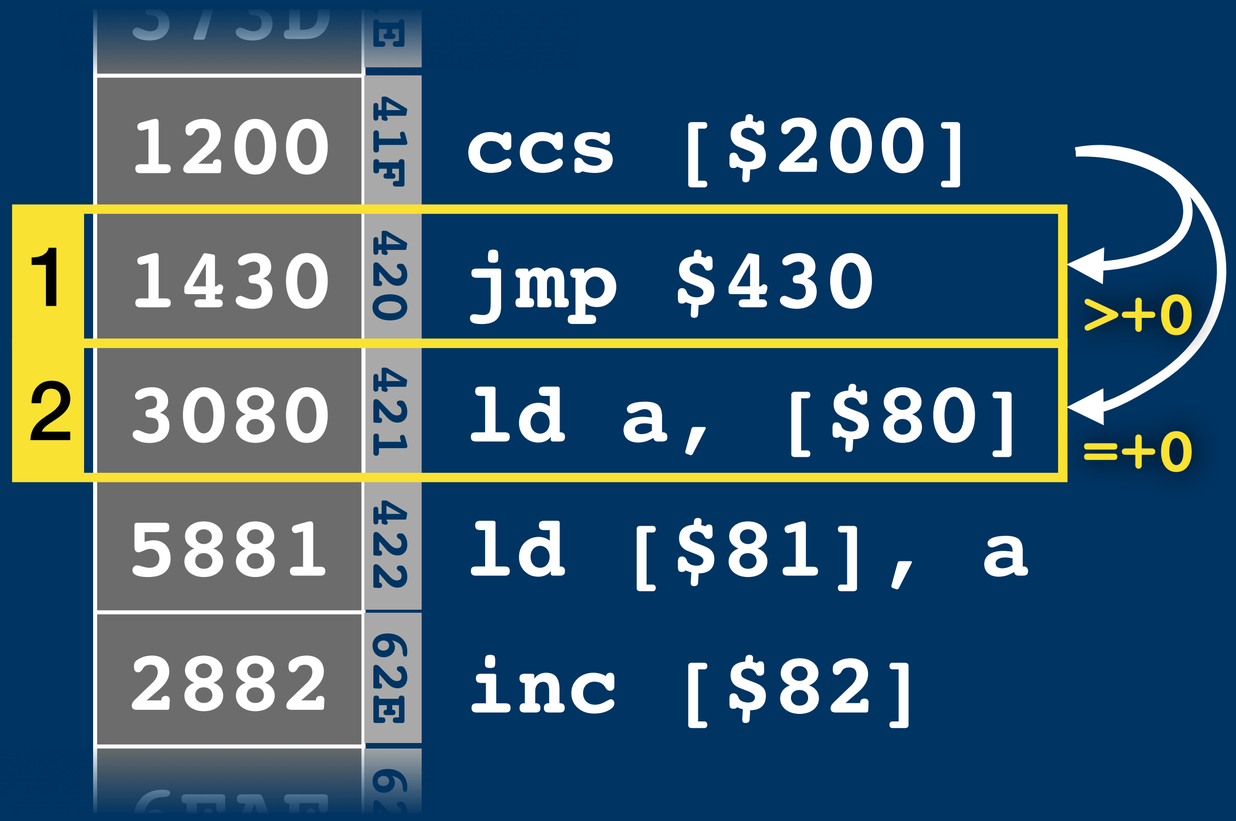

```
ccs [$200]
jmp $430
ld a, [$80]
ld [$81], a
inc [$82]
```

Conditional Jumps

Count, Compare and Skip

ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
     PC ← PC
if M[k]=+0:
     PC ← PC+1
if M[k]<-0:
     PC ← PC+2
if M[k]=-0:
     PC ← PC+3
```



Conditional Jumps

Count, Compare and Skip

ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
    PC ← PC
if M[k]=+0:
    PC ← PC+1
if M[k]<-0:
    PC ← PC+2
if M[k]=-0:
    PC ← PC+3
```

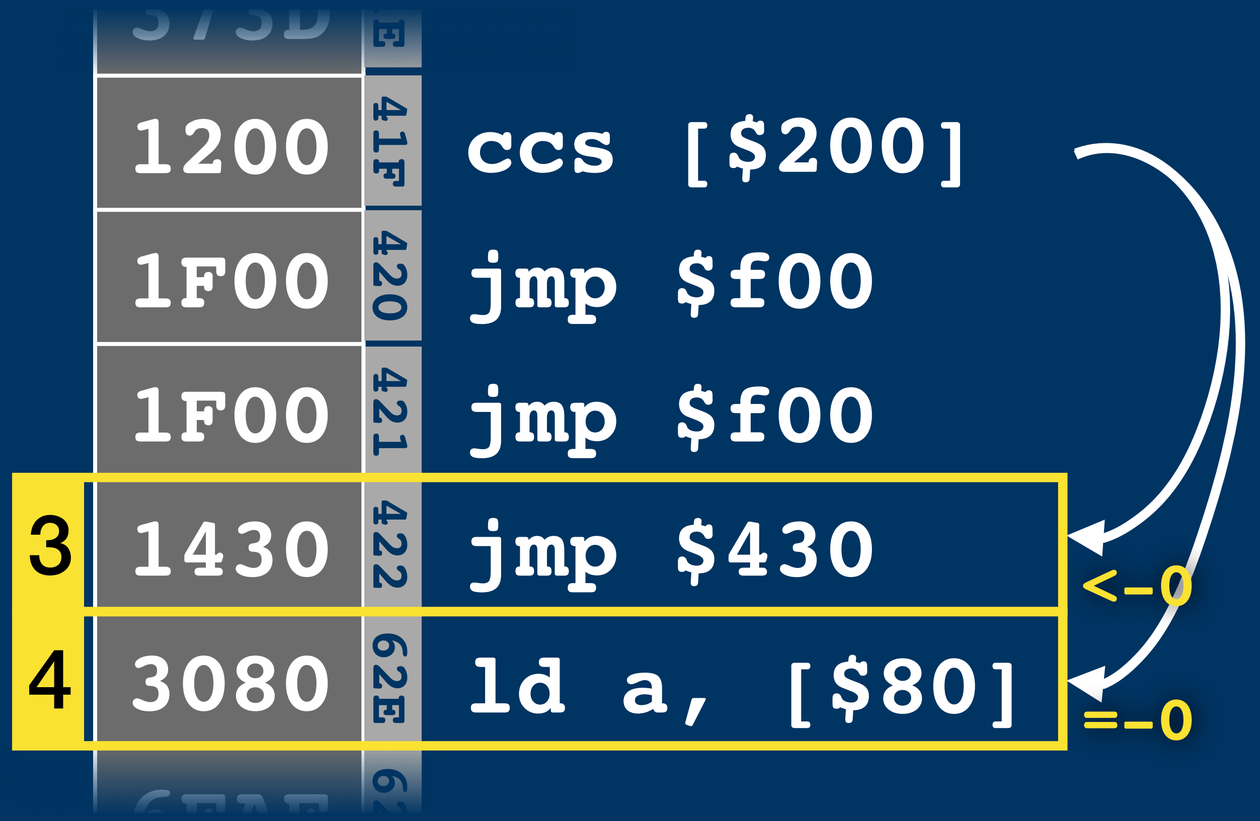
1200	41F	ccs [$200]
1F00	420	jmp $f00
1F00	421	jmp $f00
1430	422	jmp $430
3080	62E	ld a, [$80]

Conditional Jumps

Count, Compare and Skip

ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
     PC ← PC
if M[k]=+0:
     PC ← PC+1
if M[k]<-0:
     PC ← PC+2
if M[k]=-0:
     PC ← PC+3
```

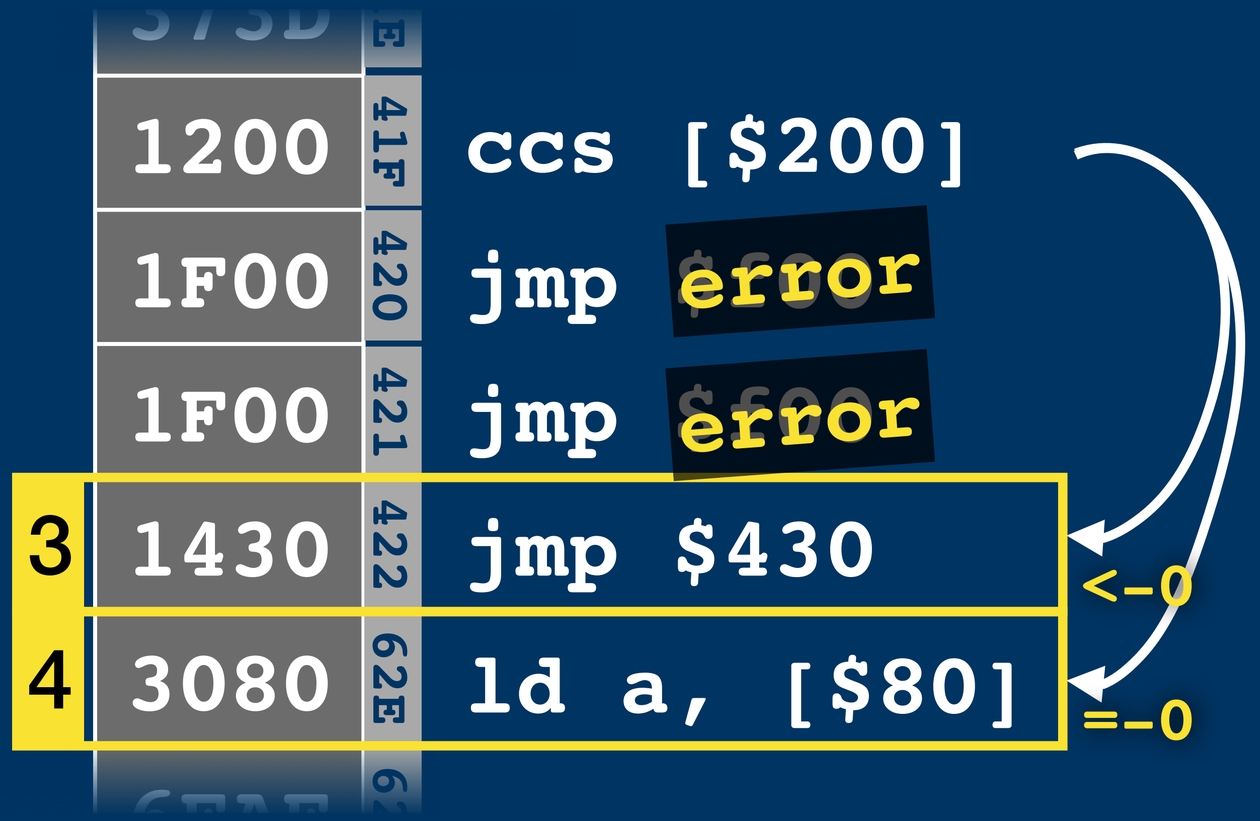


Conditional Jumps

Count, Compare and Skip

ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
     PC ← PC
if M[k]=+0:
     PC ← PC+1
if M[k]<-0:
     PC ← PC+2
if M[k]=-0:
     PC ← PC+3
```



Conditional Jumps

Count, Compare and Skip

ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
    PC ← PC
if M[k]=+0:
    PC ← PC+1
if M[k]<-0:
    PC ← PC+2
if M[k]=-0:
    PC ← PC+3
```

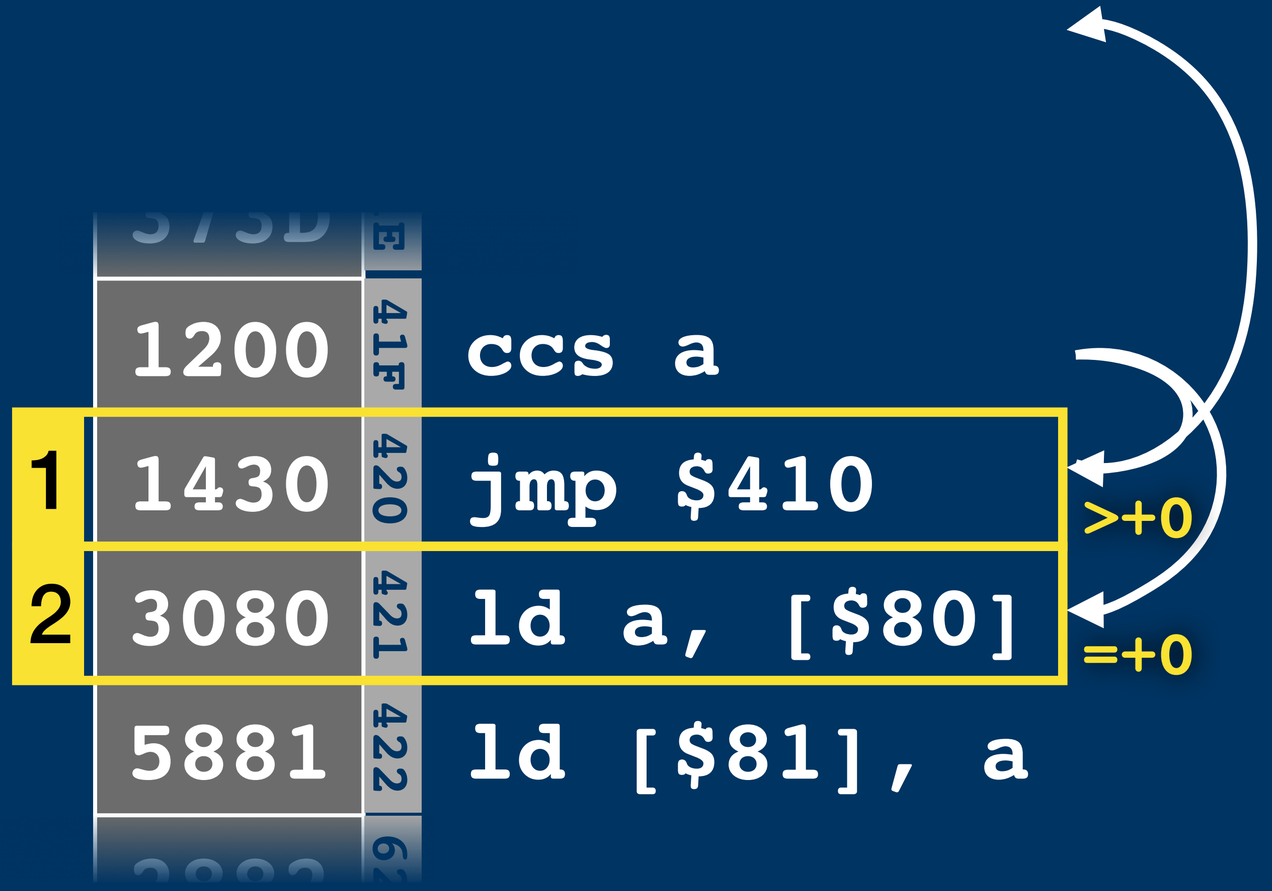
1200	41F	ccs a
1430	420	jmp $410
3080	421	ld a, [$80]
5881	422	ld [$81], a

Conditional Jumps

Count, Compare and Skip

ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
     PC ← PC
if M[k]=+0:
     PC ← PC+1
if M[k]<-0:
     PC ← PC+2
if M[k]=-0:
     PC ← PC+3
```



Call

call $e80

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	041F
BB	0000
0	0000

Call		
	call k	TC k
	0000+k	1
	LR ← PC | PC ← k	

Call

call $e80

Call	call k	TC k
	0000+k	1
	LR ← PC | PC ← k	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	041F
BB	0000
0	0000

PC →	3100	41F	ld a, [$100]
	0E80	420	call $e80
	3103	421	ld a, [$103]
	5880	E80	ld [$80], a
	0002	E81	ret

Call

call $e80

Call		
	call k	TC k
	0000+k	1
	LR ← PC | PC ← k	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0420
BB	0000
0	0000

PC →

3100	41F	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]

5880	E80	ld [$80], a
0002	E81	ret

Call

call $e80

Call		
	call k	TC k
	0000+k	1
	LR ← PC | PC ← k	

A	0007
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

PC →

3100	41F	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]
5880	E80	ld [$80], a
0002	E81	ret

Call

call $e80

Call		
	call k	TC k
	0000+k	1
	LR ← PC | PC ← k	

A	0007
B	0F00
LR	0421
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

PC →

3100	41F	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]

5880	E80	ld [$80], a
0002	E81	ret

Call

call $e80

Call		
	call k	TC k
	0000+k	1
	LR ← PC | PC ← k	

A	0007
B	0F00
LR	0421
EB	0000
FB	0000
PC	0E80
BB	0000
0	0000

PC →

3100	41F	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]

5880	E80	ld [$80], a
0002	E81	ret

Call

call $e80

Call		
	call k	TC k
	0000+k	1
	LR ← PC | PC ← k	

A	0007
B	0F00
LR	0421
EB	0000
FB	0000
PC	0E80
BB	0000
0	0000

3100	41F	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]

PC →

5880	E80	ld [$80], a
0002	E81	ret

Return

ret

Return	ret	RETURN
	0002 (alias)	1
	LR ← PC | PC ← 2	

A	0007
B	0F00
LR	0421
EB	0000
FB	0000
PC	0E80
BB	0000
0	0000

3100	41F	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]

PC →

5880	E80	ld [$80], a
0002	E81	ret

Return

ret

Return	ret	RETURN
	0002 (alias)	1
	LR ← PC | PC ← 2	

A	0007
B	0F00
LR	0421
EB	0000
FB	0000
PC	0E81
BB	0000
0	0000

3100	41F	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]

	5880	E80	ld [$80], a
PC →	0002	E81	ret

Return

ret

Return		
	ret	RETURN
	0002 (alias)	1
	LR ← PC | PC ← 2	

A	0007
B	0F00
LR	0421
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

3100	41F	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]
5880	E80	ld [$80], a
PC → 0002	E81	ret

Return

ret

Return	ret	RETURN
	0002 (alias)	1
	LR ← PC | PC ← 2	

A	0007
B	0F00
LR	0421
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

	3100	41F	ld a, [$100]
	0E80	420	call $e80
PC →	3103	421	ld a, [$103]
	5880	E80	ld [$80], a
	0002	E81	ret

Exchange LR

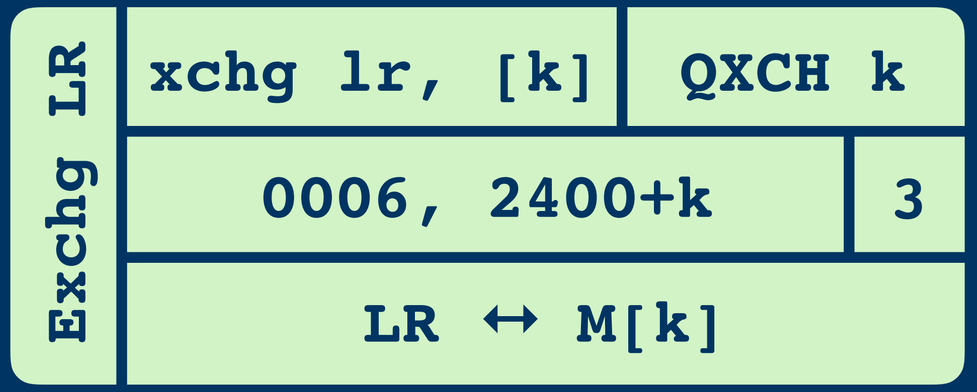
Exchg LR		
	`xchg lr, [k]`	`QXCH k`
	`0006, 2400+k`	`3`
	`LR ↔ M[k]`	

Exchange LR

Exchg LR	xchg lr, [k]	QXCH k
	0006, 2400+k	3
	LR ↔ M[k]	

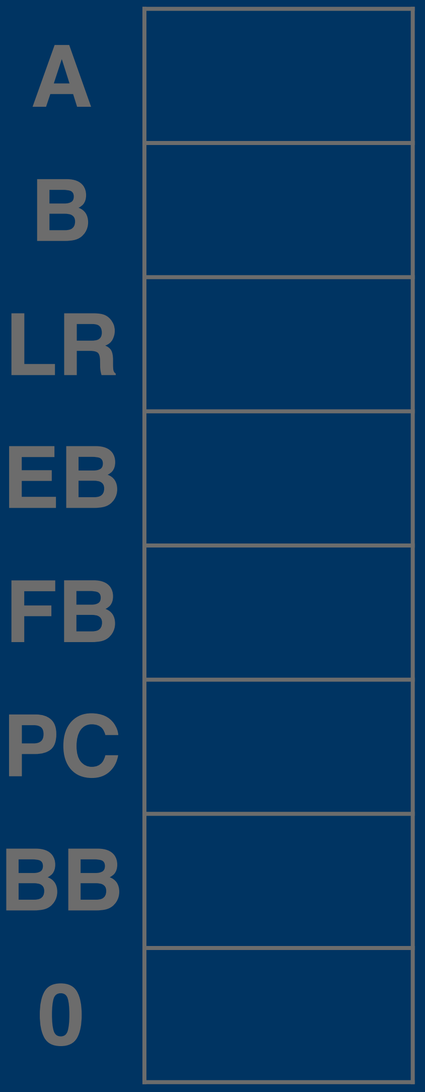
```
call $ee0

ret
```

Exchange LR

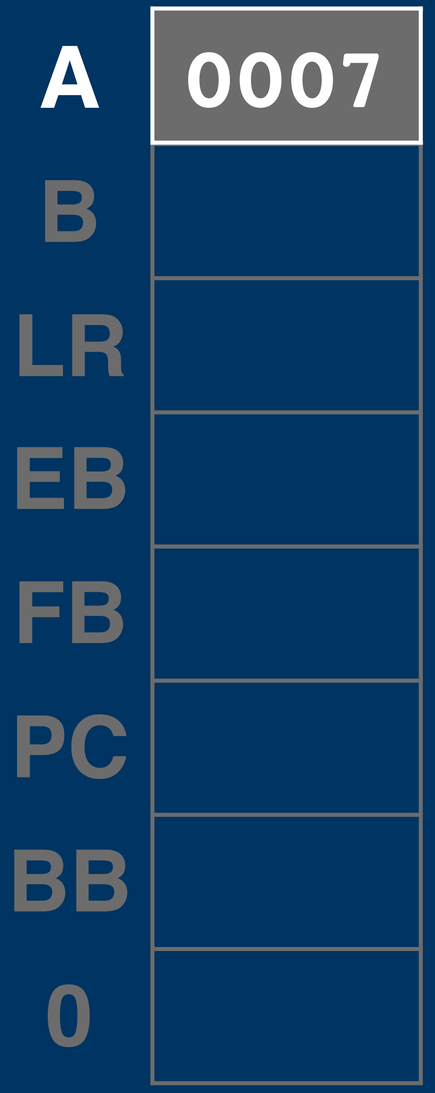



```
xchg lr, [$4ff]
call $ee0
xchg lr, [$4ff]
ret
```

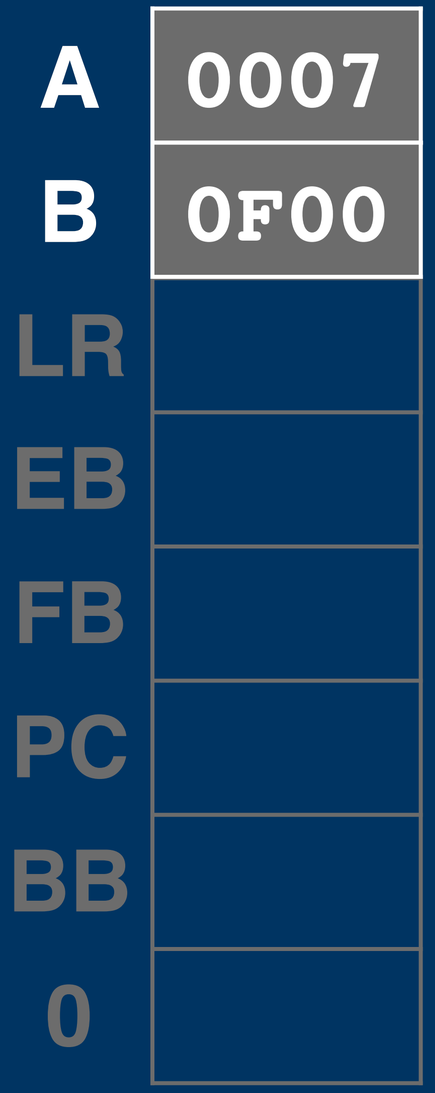
Registers



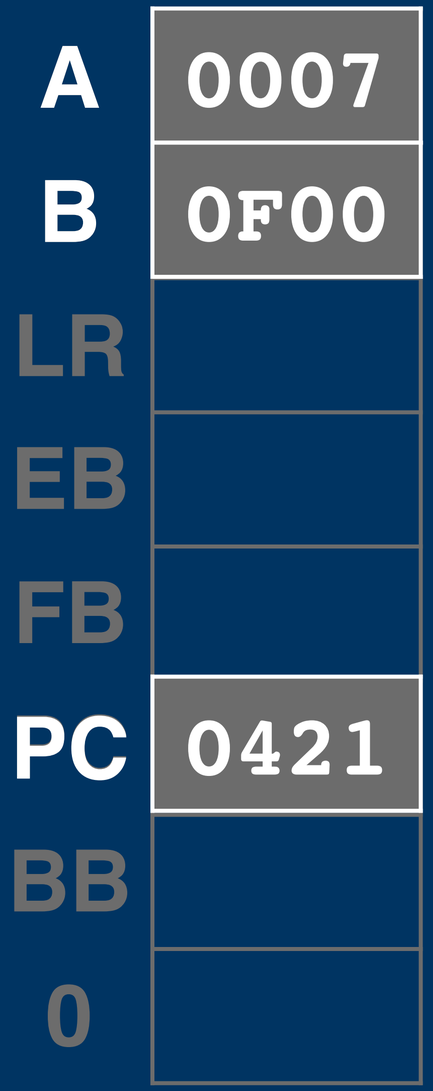
Registers



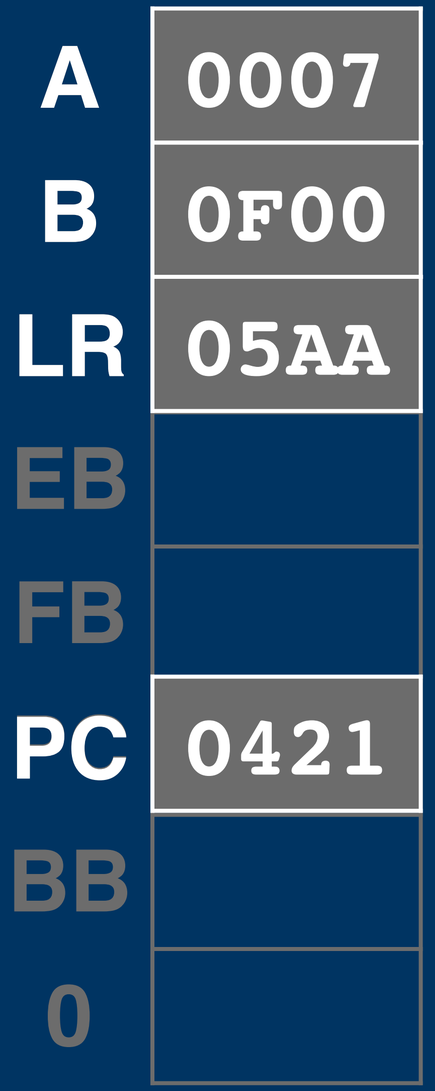
Registers



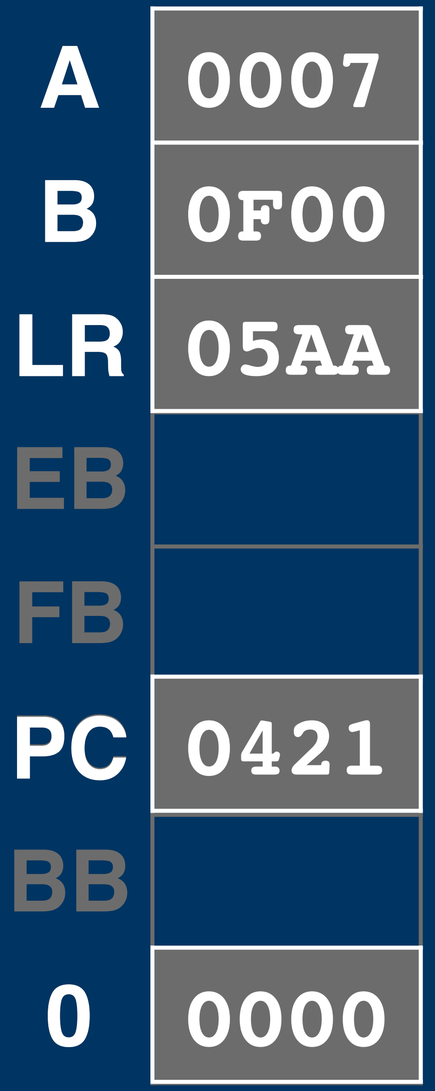
Registers



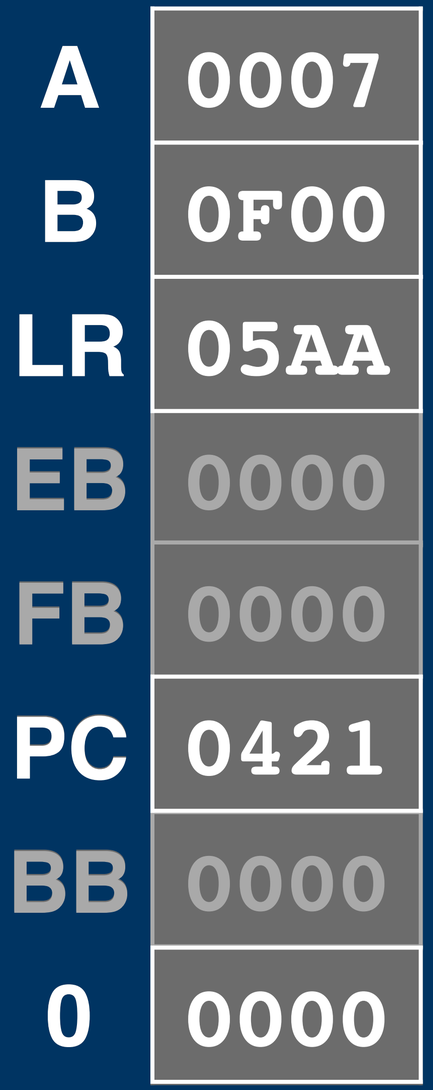
Registers



Registers

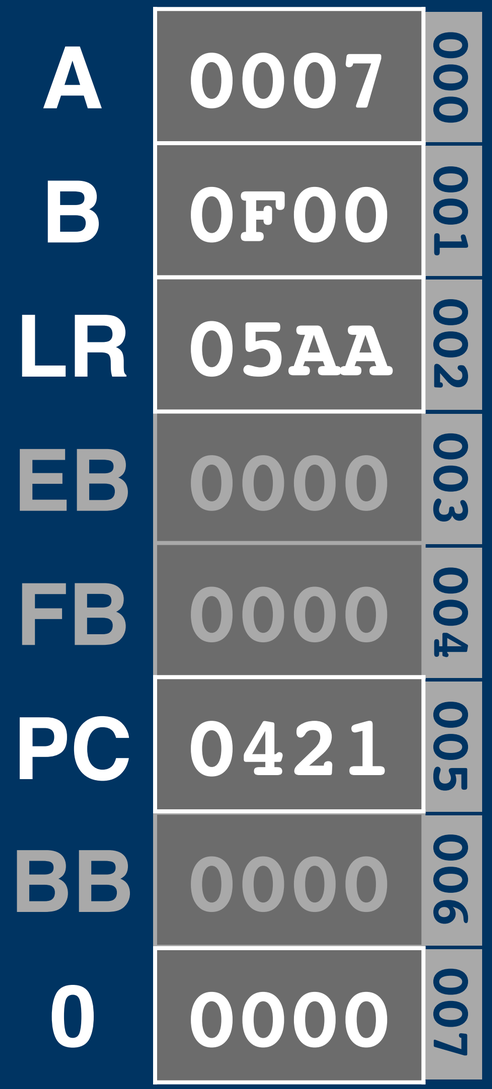


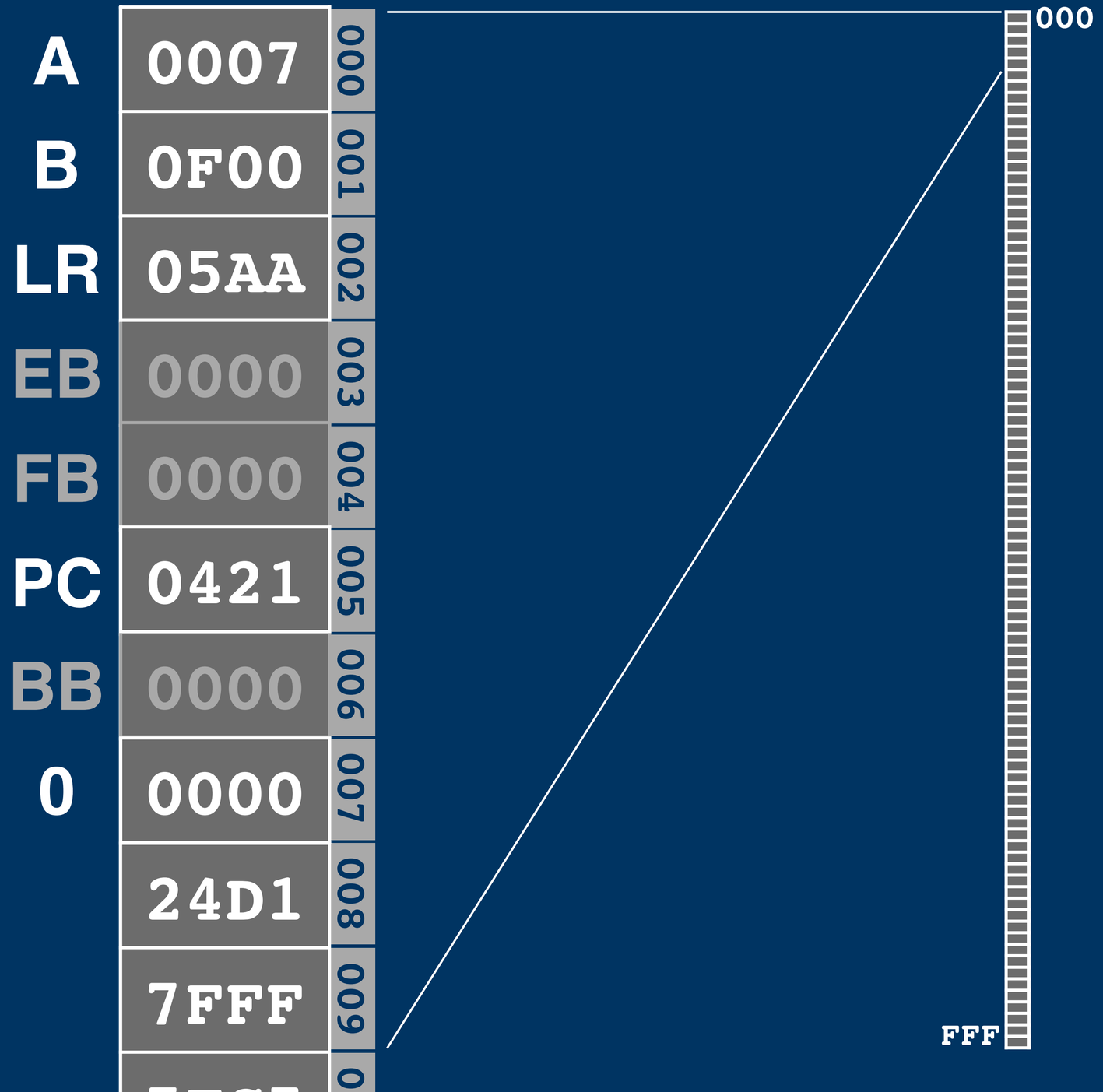
Registers

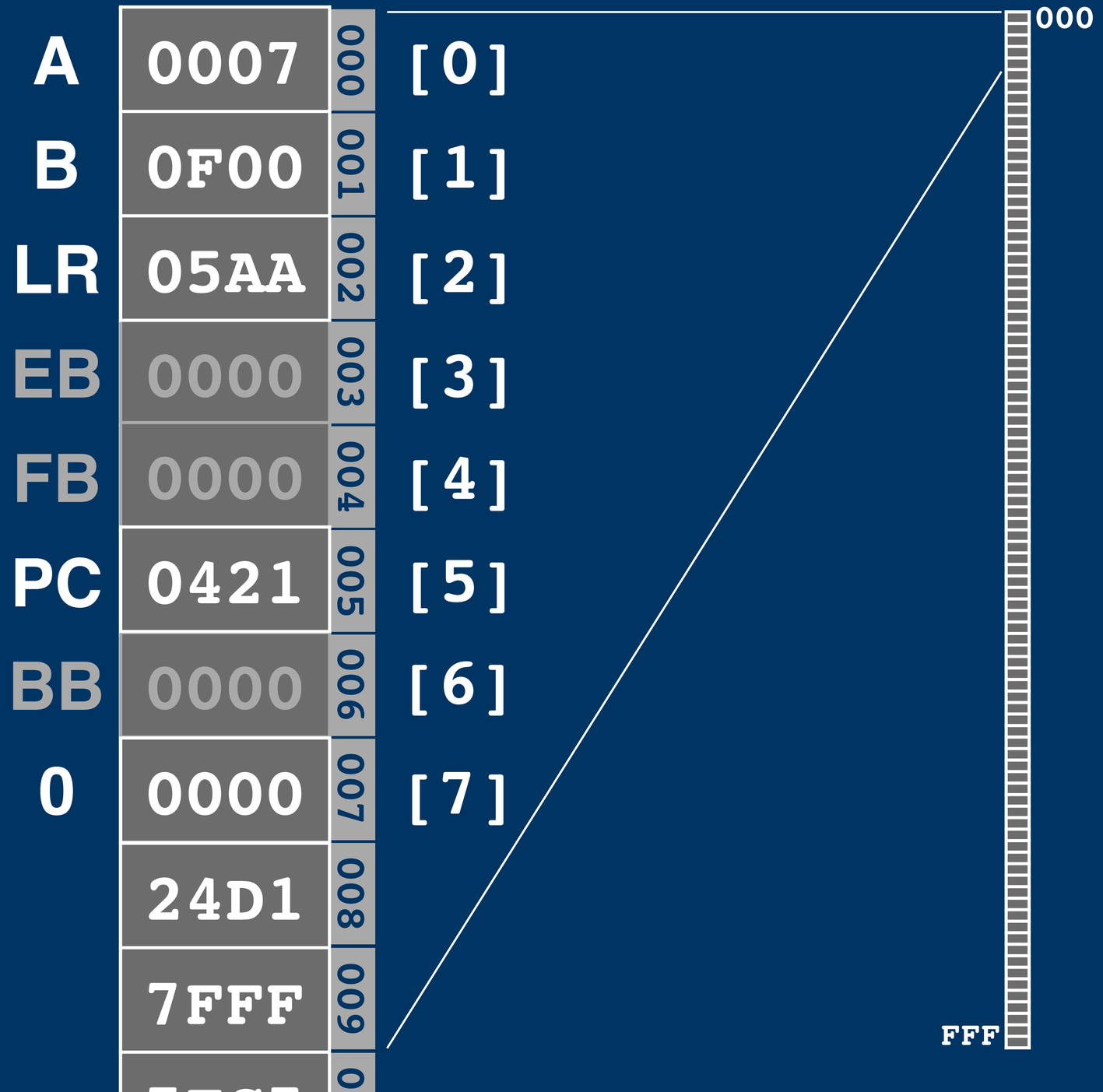


Registers


A
0007
000
B
0F00
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007

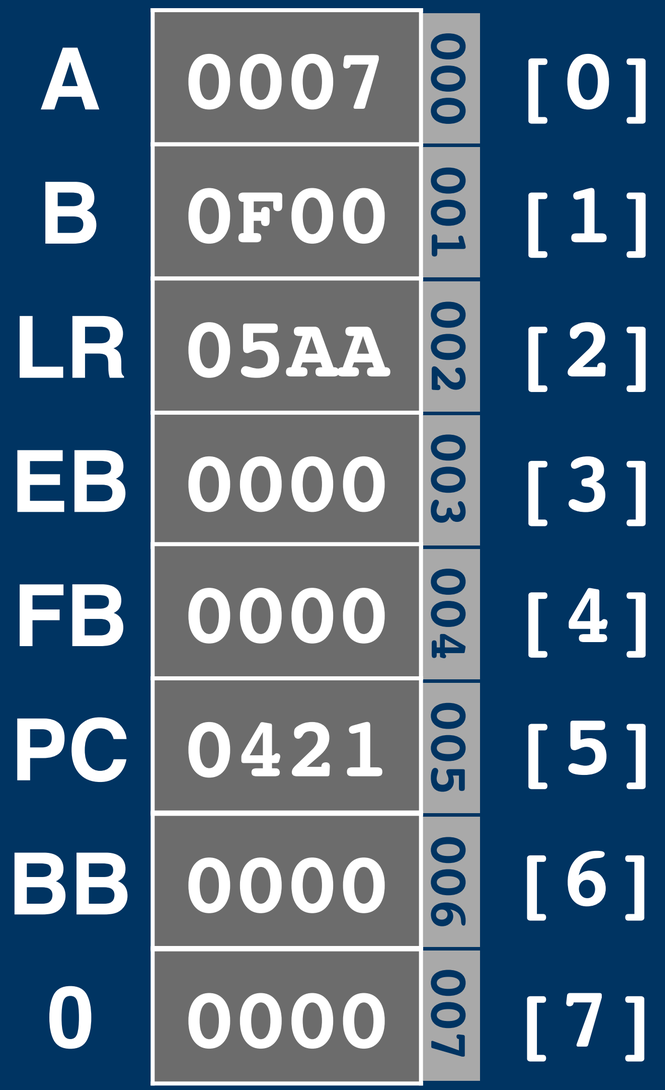

A
0007
000
B
0F00
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007


A
0007
000
B
0F00
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007
24D1
008
7FFF
009
000
FFF


A
0007
000
[0]
B
0F00
001
[1]
LR
05AA
002
[2]
EB
0000
003
[3]
FB
0000
004
[4]
PC
0421
005
[5]
BB
0000
006
[6]
0
0000
007
[7]
24D1
008
7FFF
009
000
FFF

Memory-Mapped Registers

Memory-Mapped Registers

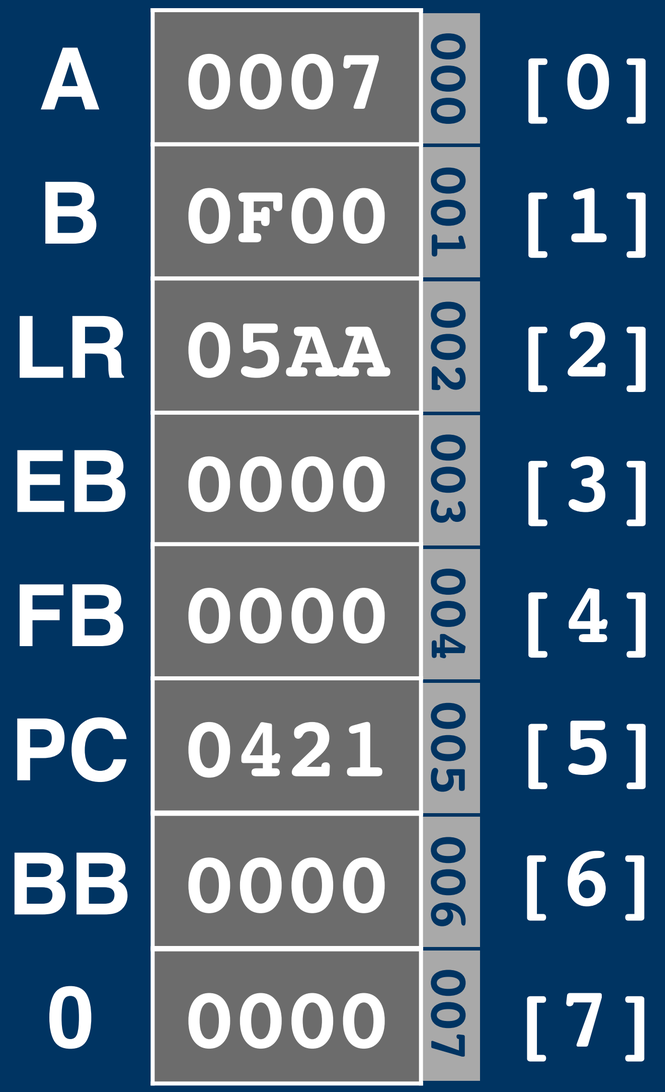



```
ld a, b
```

Memory-Mapped Registers

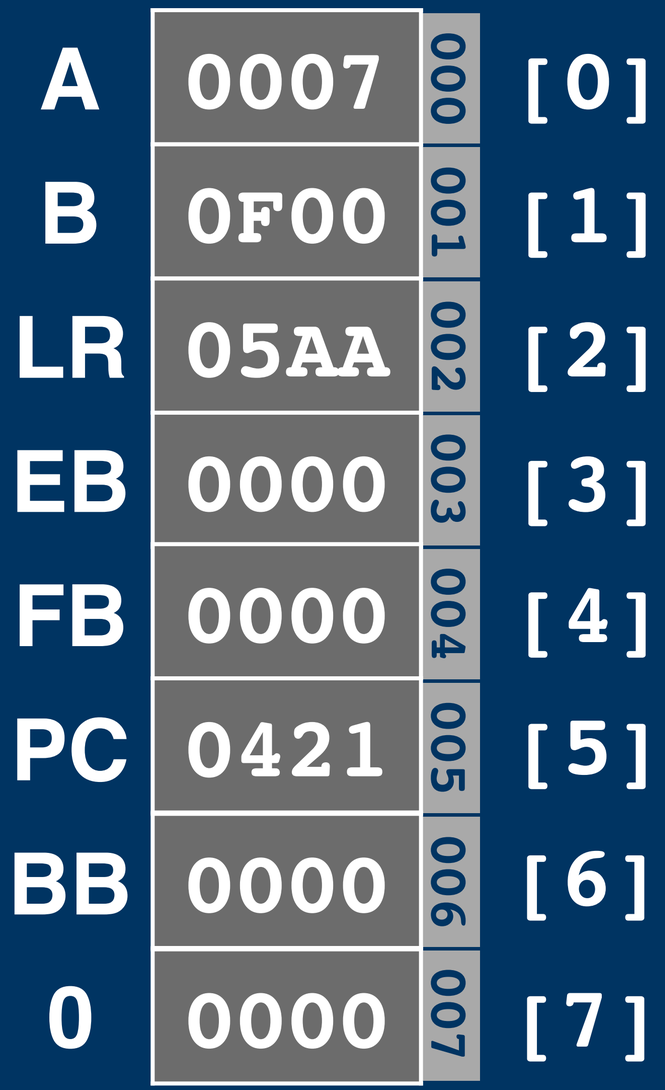

```
ld a, b        ≡    ld a, [1]
```

Memory-Mapped Registers



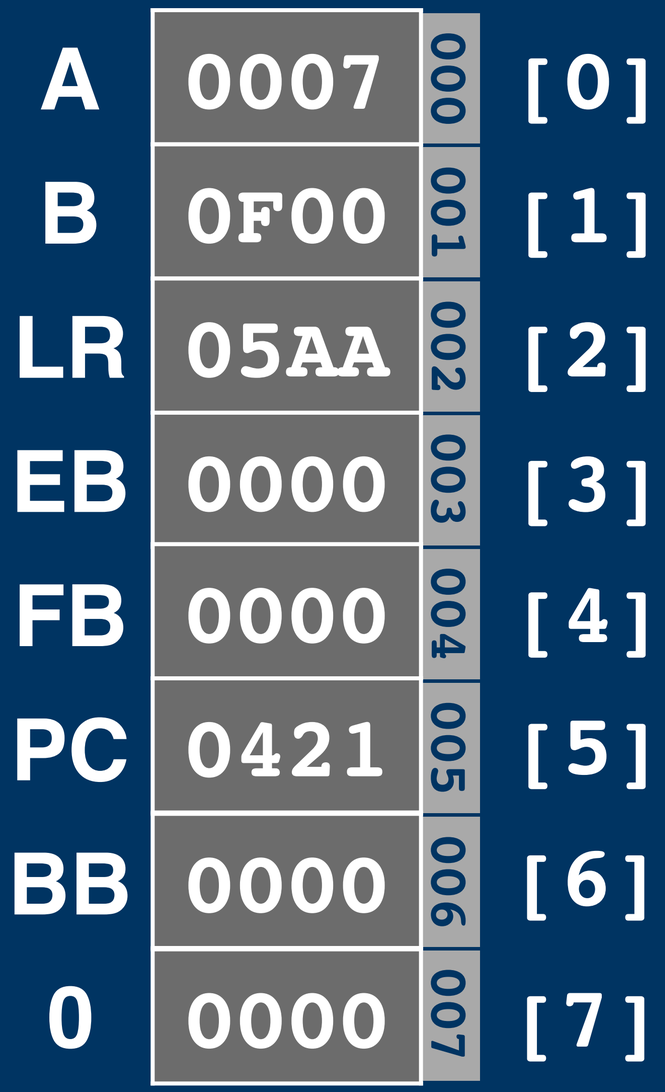

```
ld a, b        ≡    ld a, [1]

ld a, #0
```

Memory-Mapped Registers



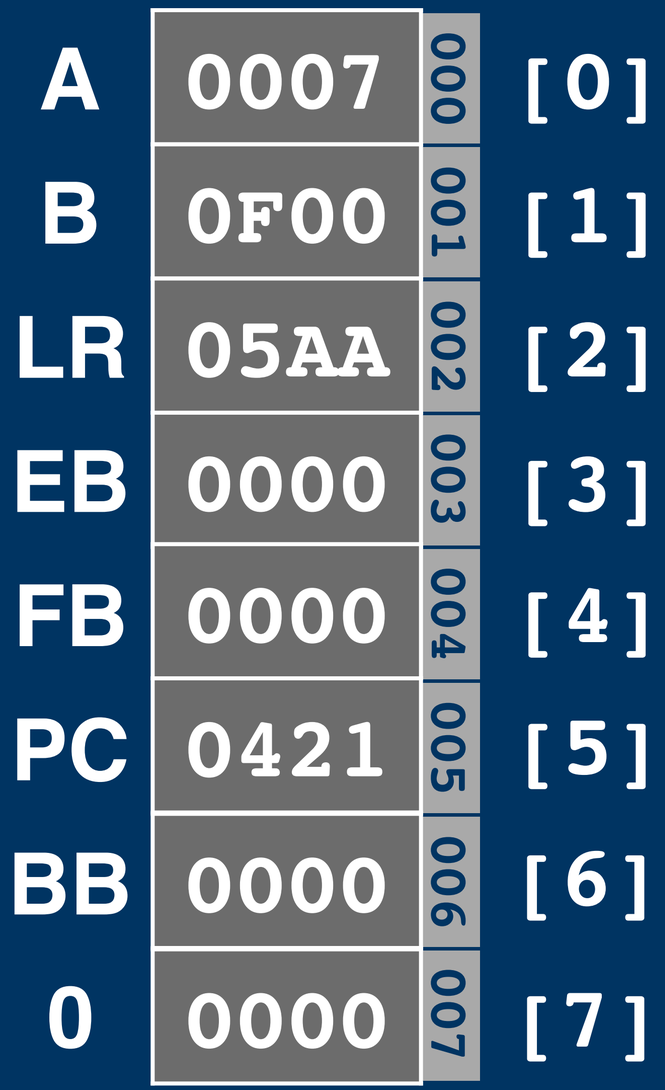

```
ld a, b        ≡    ld a, [1]

ld a, #0       ≡    ld a, [7]
```

Memory-Mapped Registers



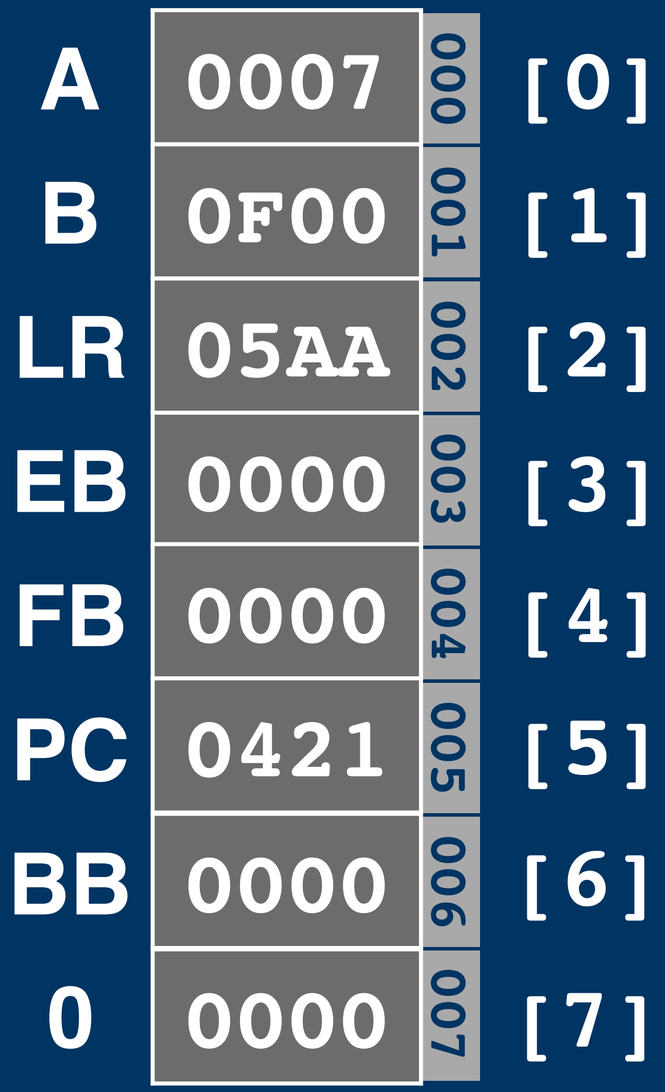

```
ld a, b        ≡    ld a, [1]

ld a, #0       ≡    ld a, [7]

inc a
```

Memory-Mapped Registers



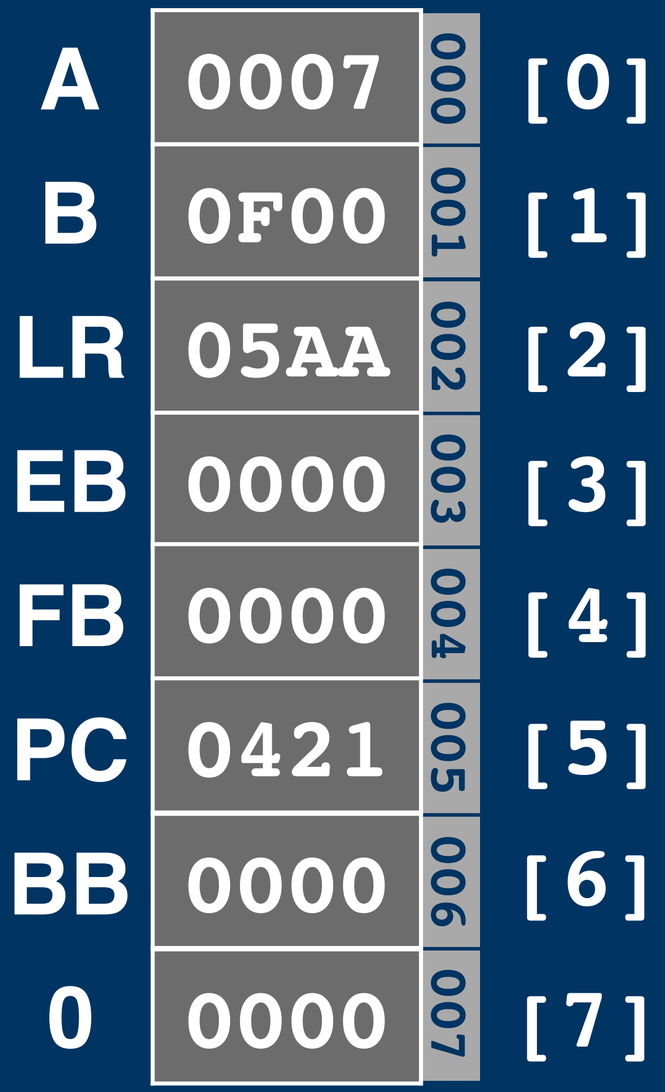

```
ld a, b        ≡    ld a, [1]

ld a, #0       ≡    ld a, [7]

inc a          ≡    inc [0]
```

Memory-Mapped Registers



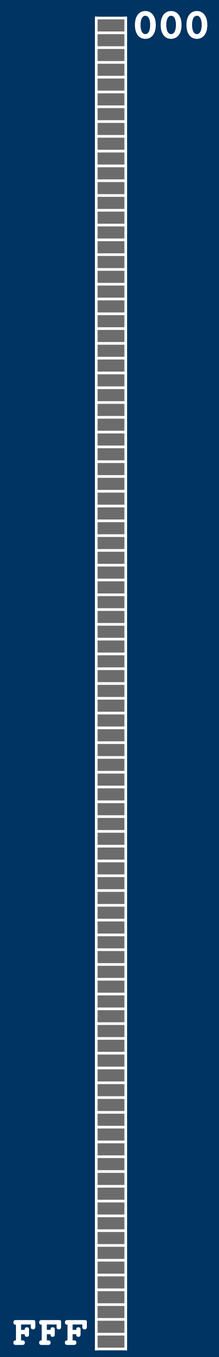

```
ld a, b        ≡    ld a, [1]

ld a, #0       ≡    ld a, [7]

inc a          ≡    inc [0]

ld a, [b]
```

Memory-Mapped Registers




```
ld a, b      ≡   ld a, [1]

ld a, #0     ≡   ld a, [7]

inc a        ≡   inc [0]

ld a, [b]    ≡   ld a, [[1]]
```

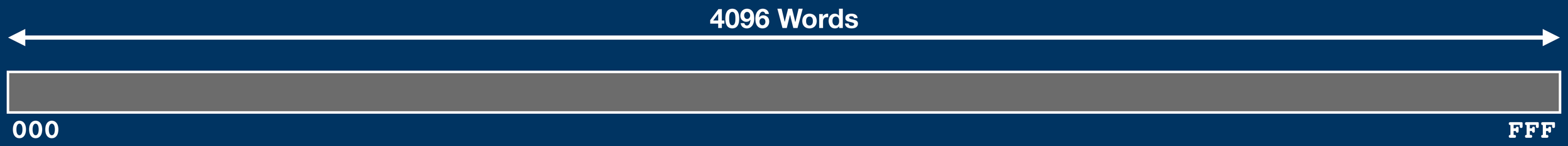
Memory Map



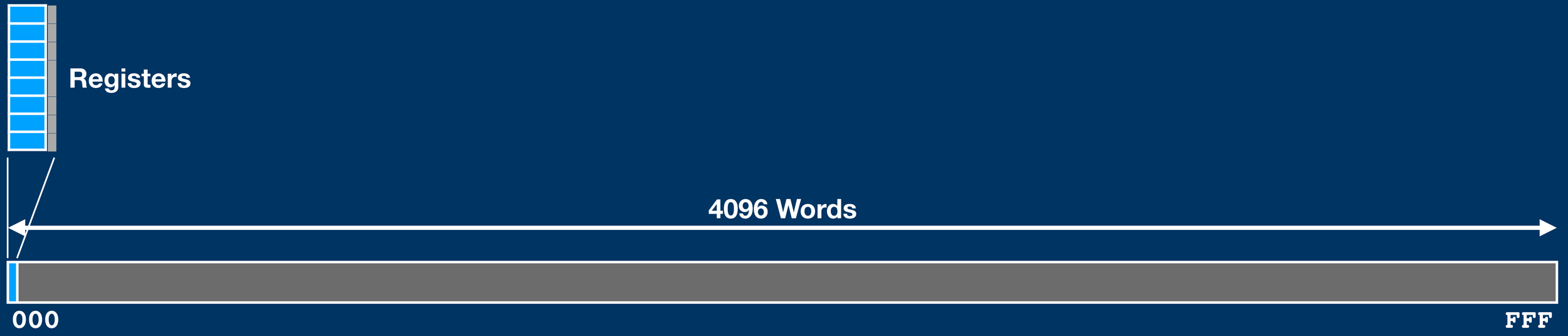
Memory Map

Memory Map

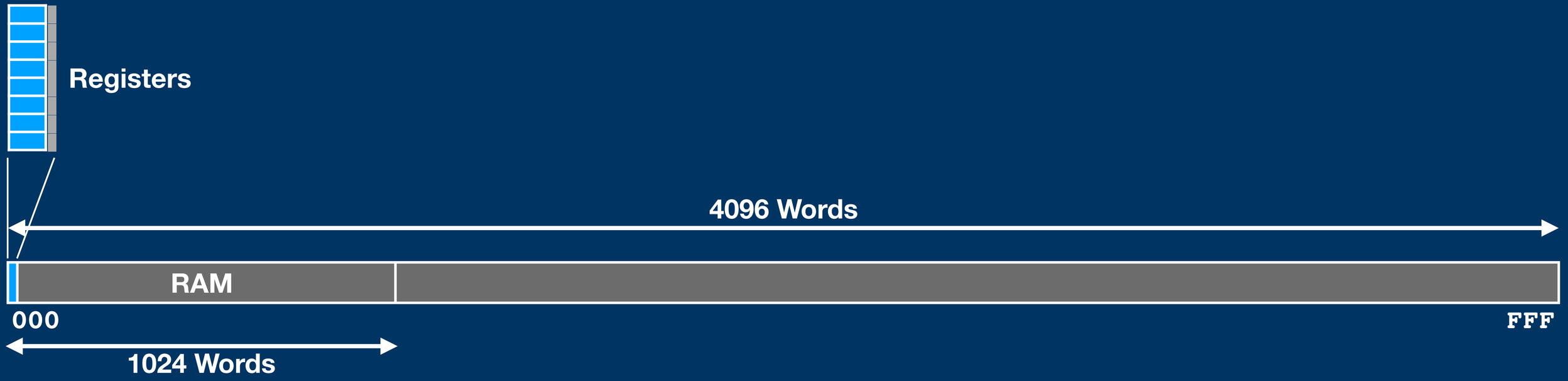
Memory Map



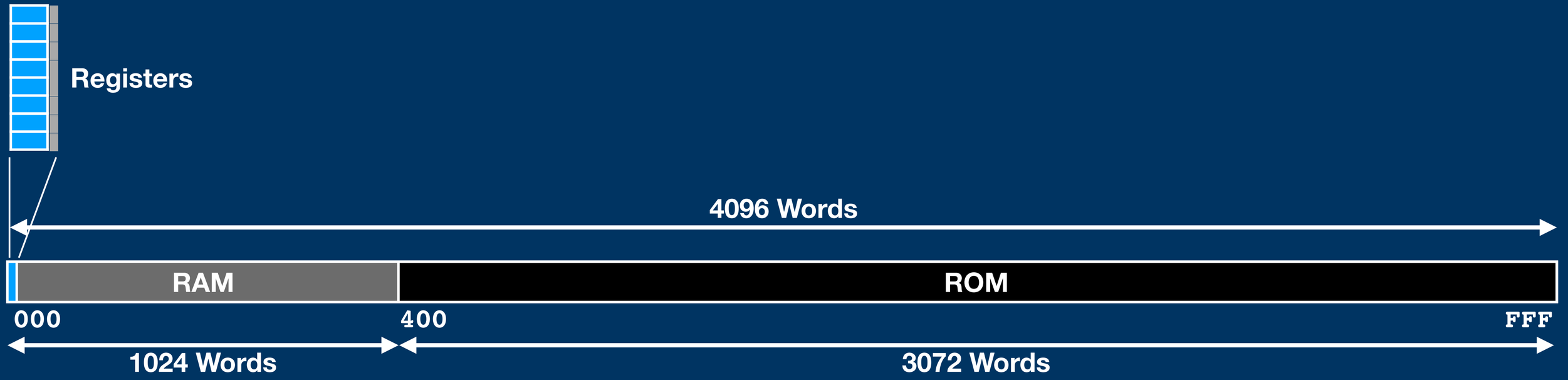
Memory Map



Memory Map

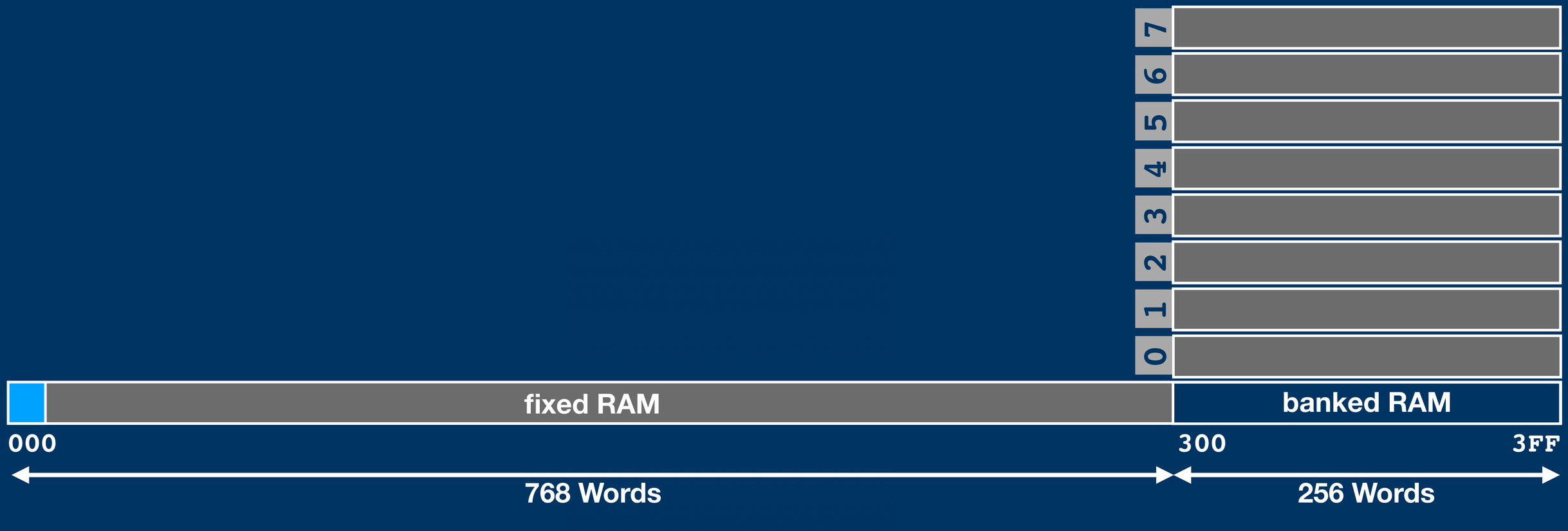


Memory Map


Registers
4096 Words
RAM
ROM
000
400
FFF
1024 Words
3072 Words

Memory Map: RAM

Memory Map: RAM

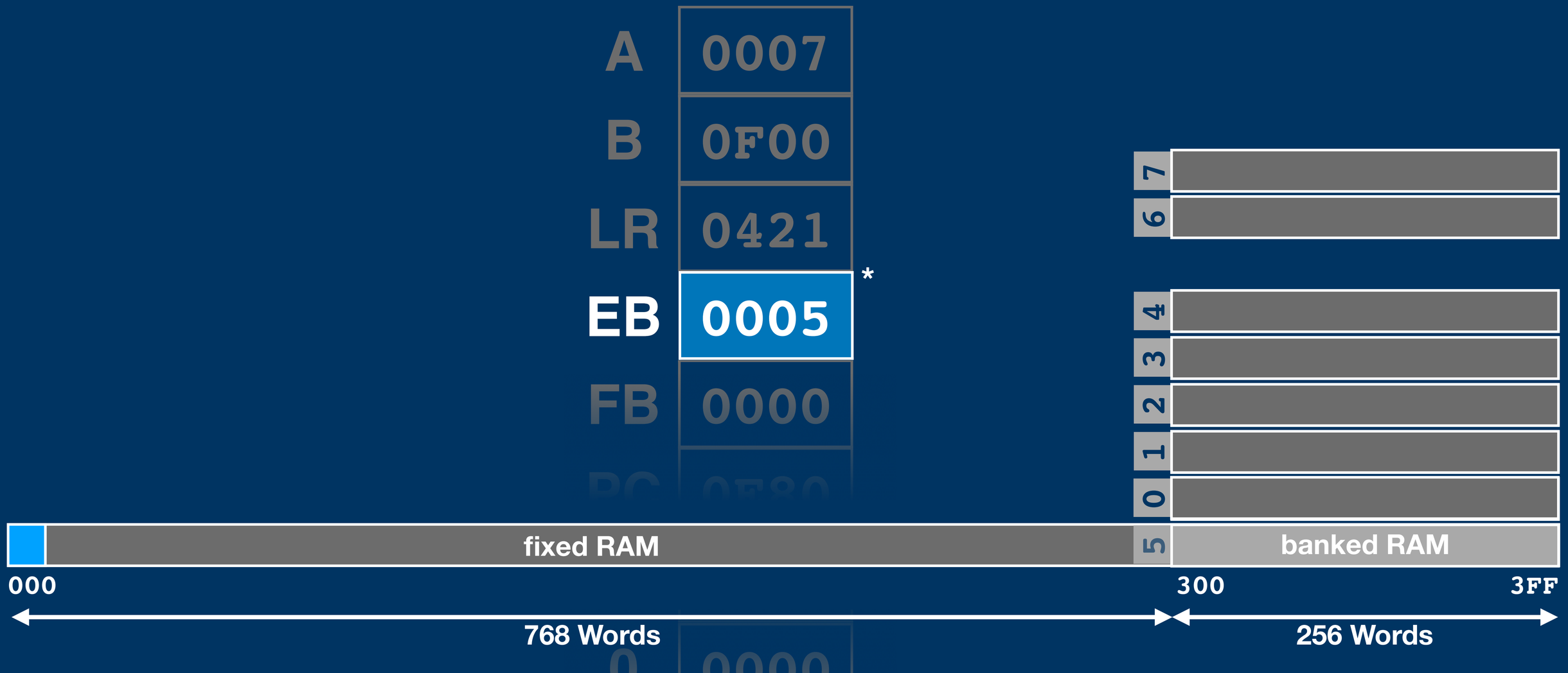


Memory Map: RAM

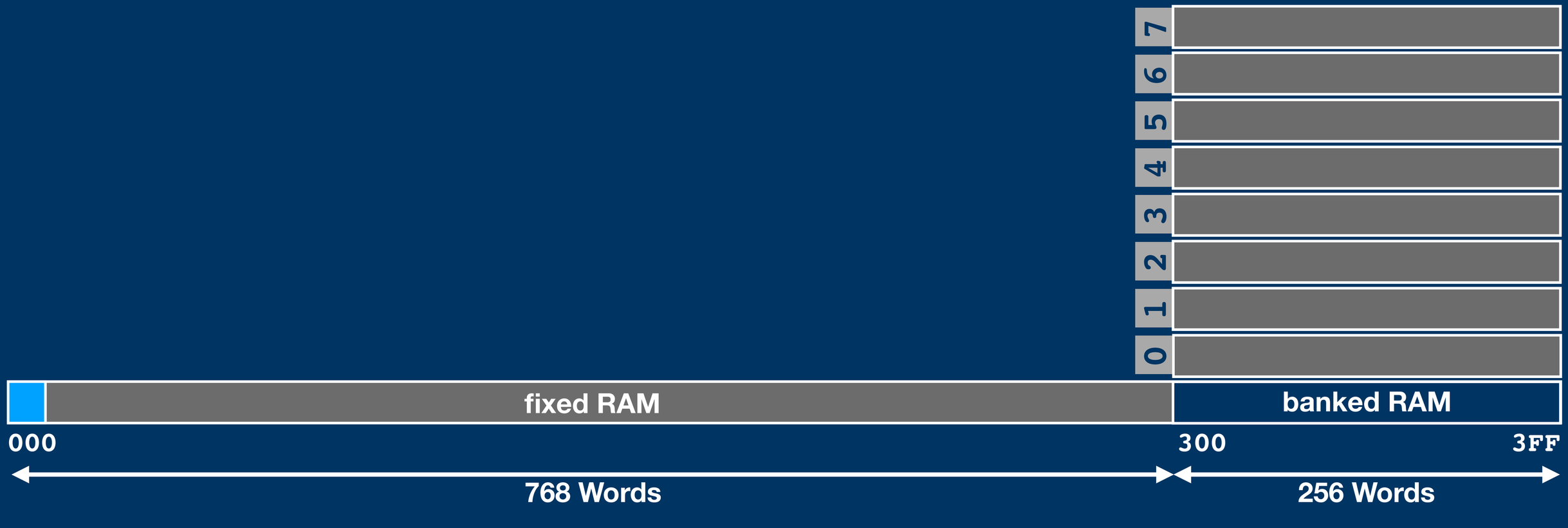
A	0007
B	0F00
LR	0421
EB	**0000**
FB	0000

Memory Map: RAM

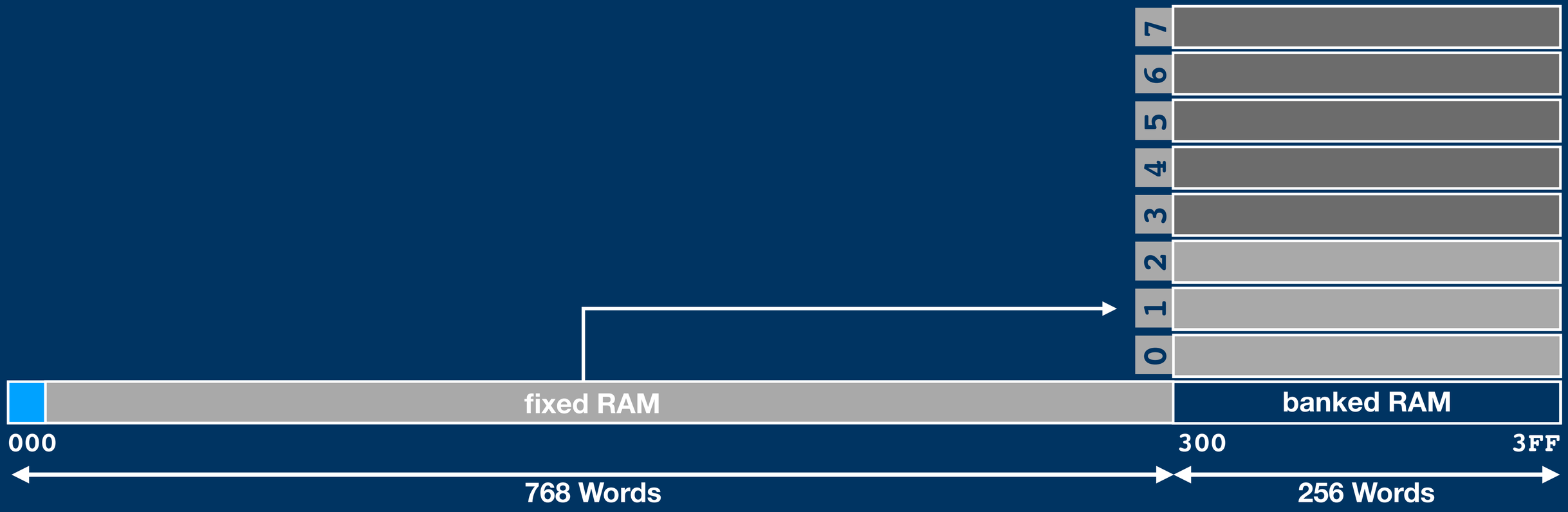
* the 3 bits are actually << 8, so EB will contain 0500



Memory Map: RAM



Memory Map: RAM

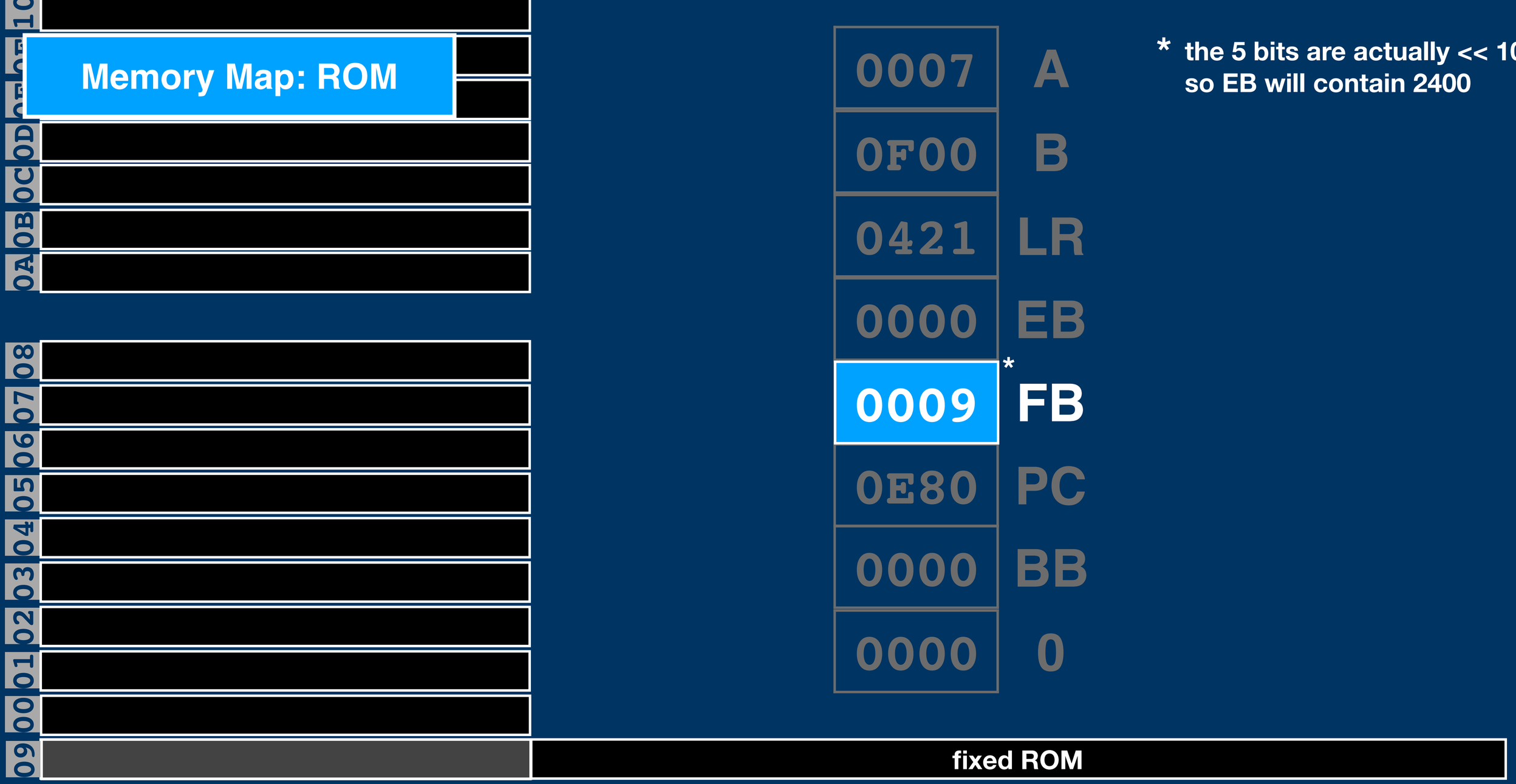


Memory Map: RAM

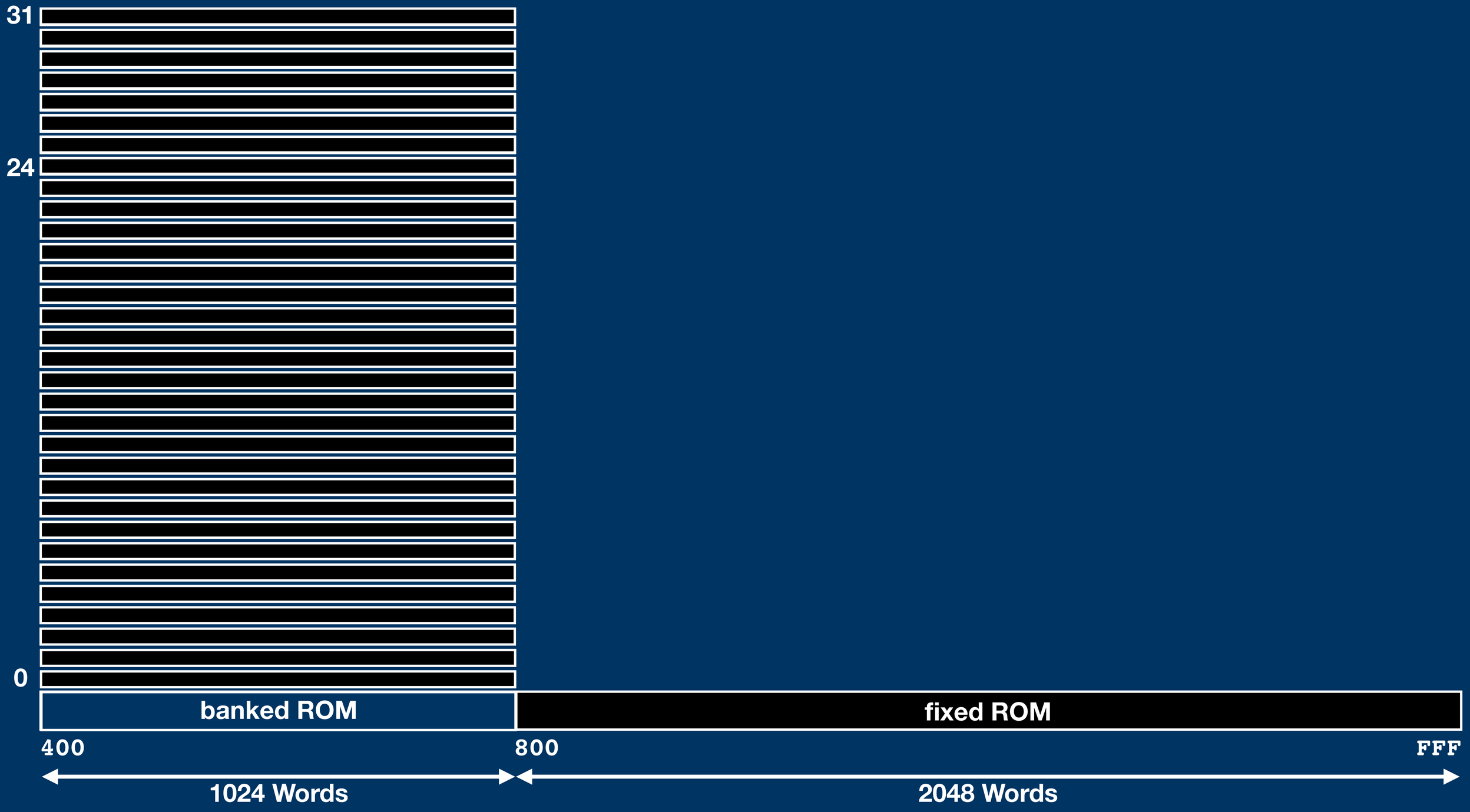
Memory Map

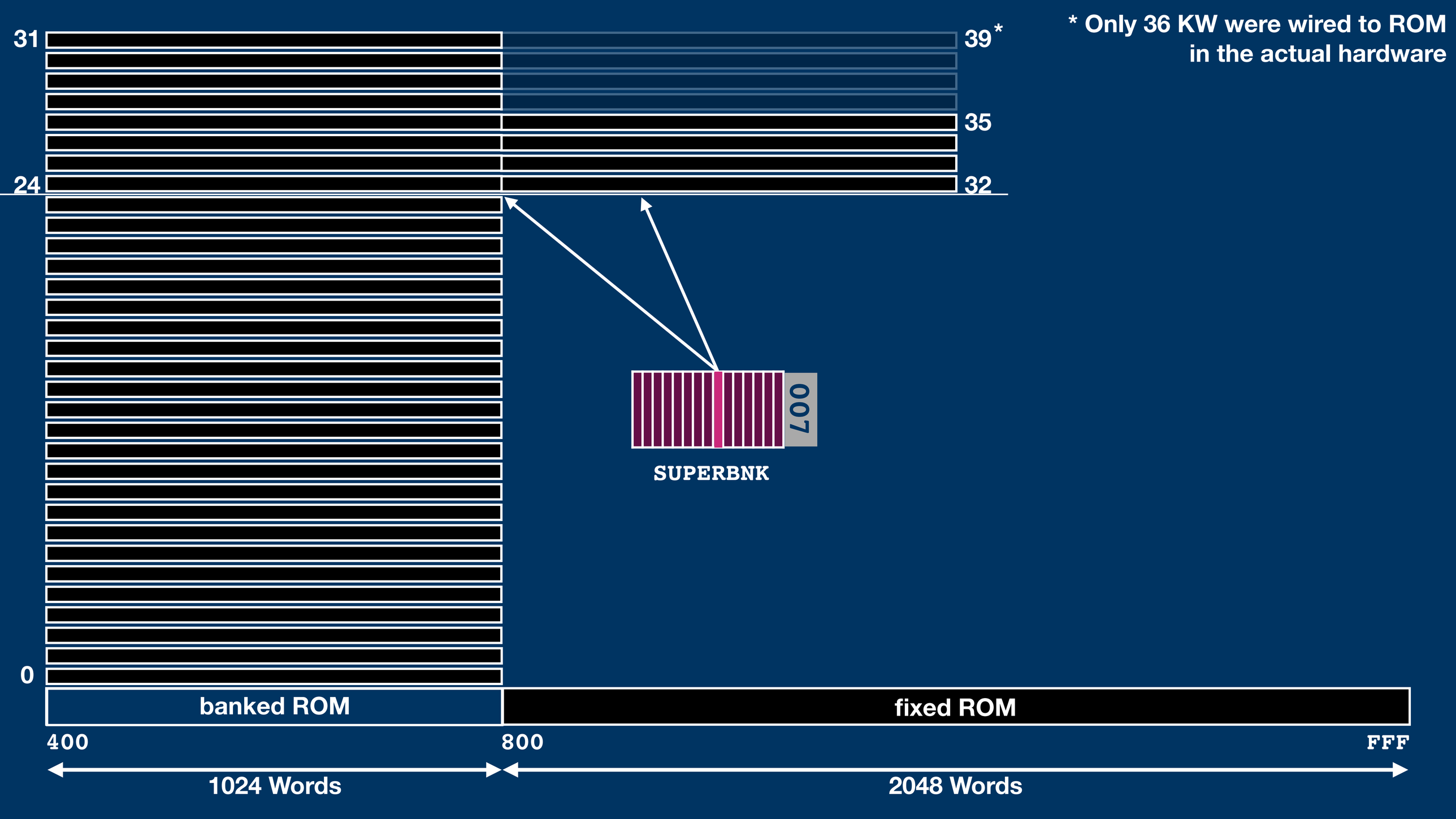
Memory Map: ROM

Memory Map: ROM

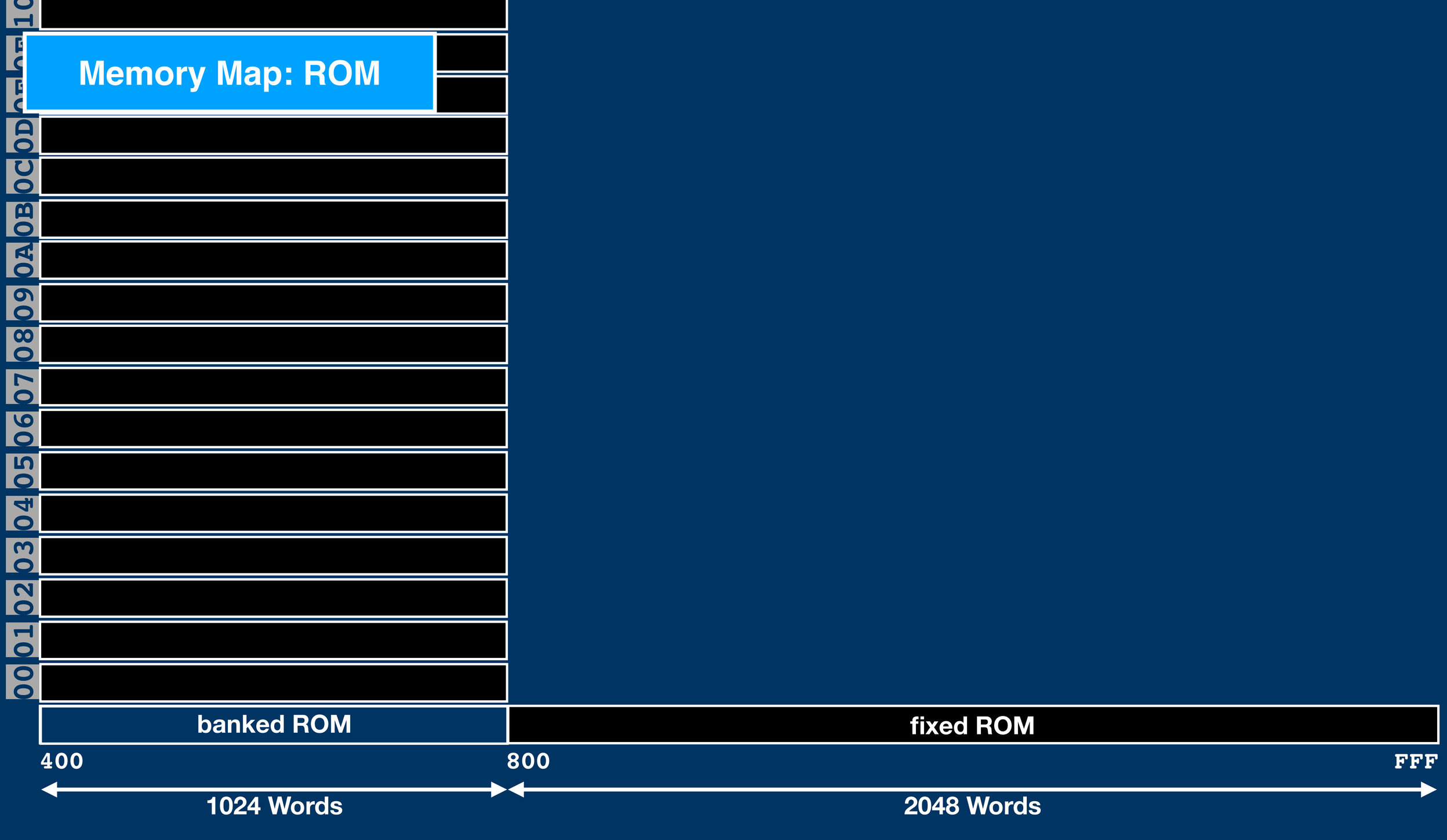


* the 5 bits are actually << 10, so EB will contain 2400

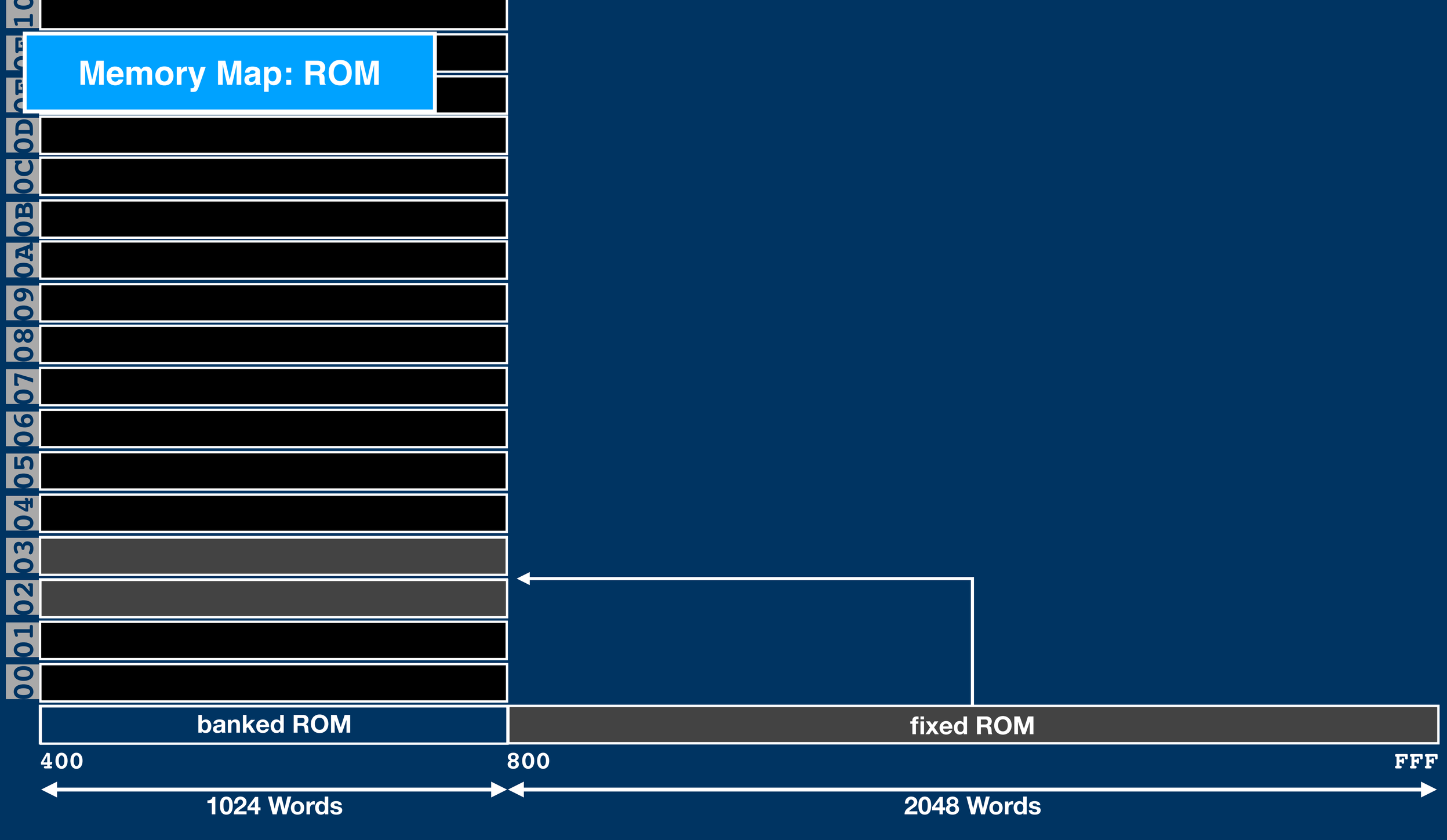

31
24
0
banked ROM
fixed ROM
400
800
FFF
1024 Words
2048 Words


* Only 36 KW were wired to ROM in the actual hardware
31
39*
35
24
32
007
SUPERBNK
0
banked ROM
fixed ROM
400
800
FFF
1024 Words
2048 Words

Memory Map: ROM



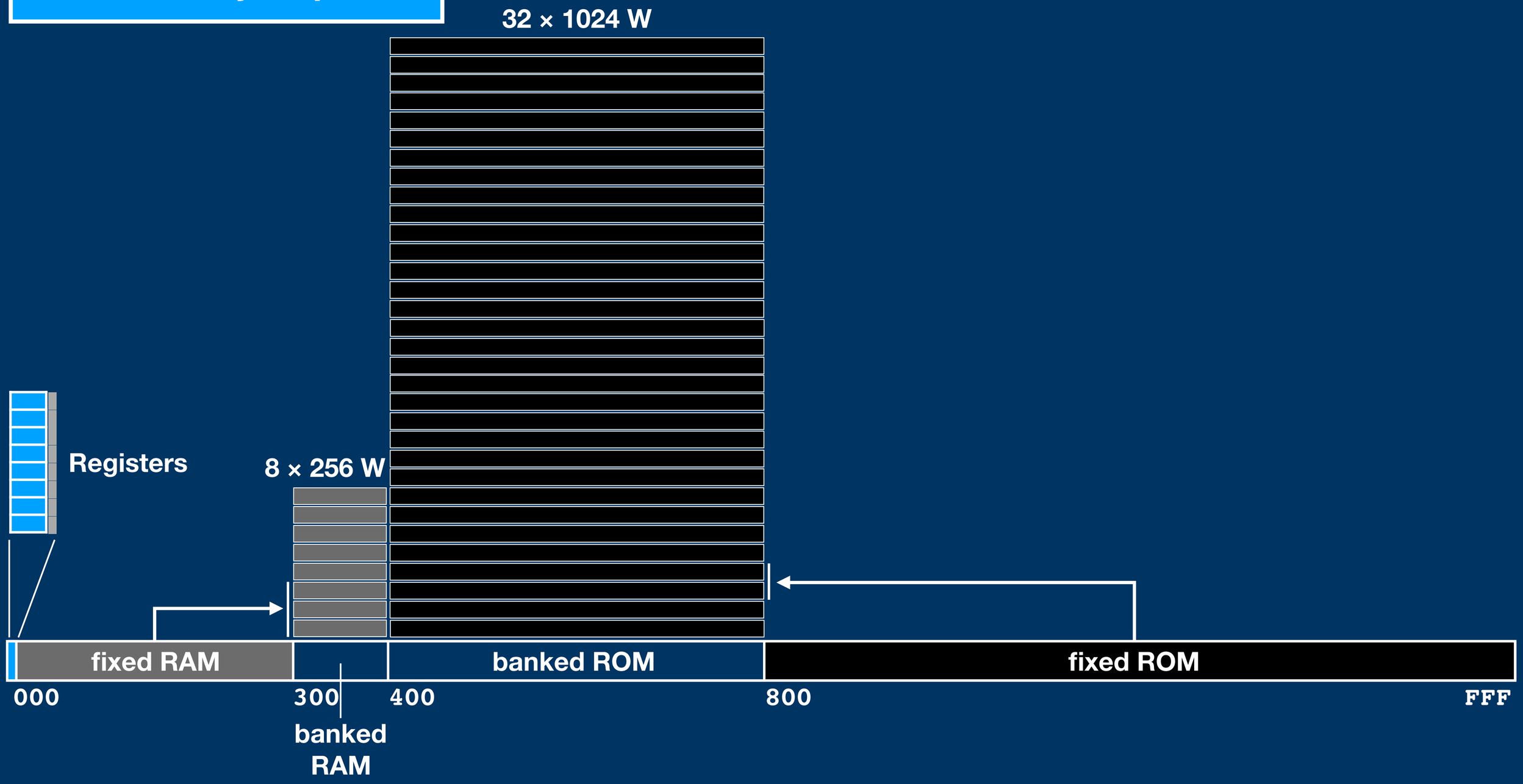
Memory Map: ROM



Memory Map: ROM

Memory Map

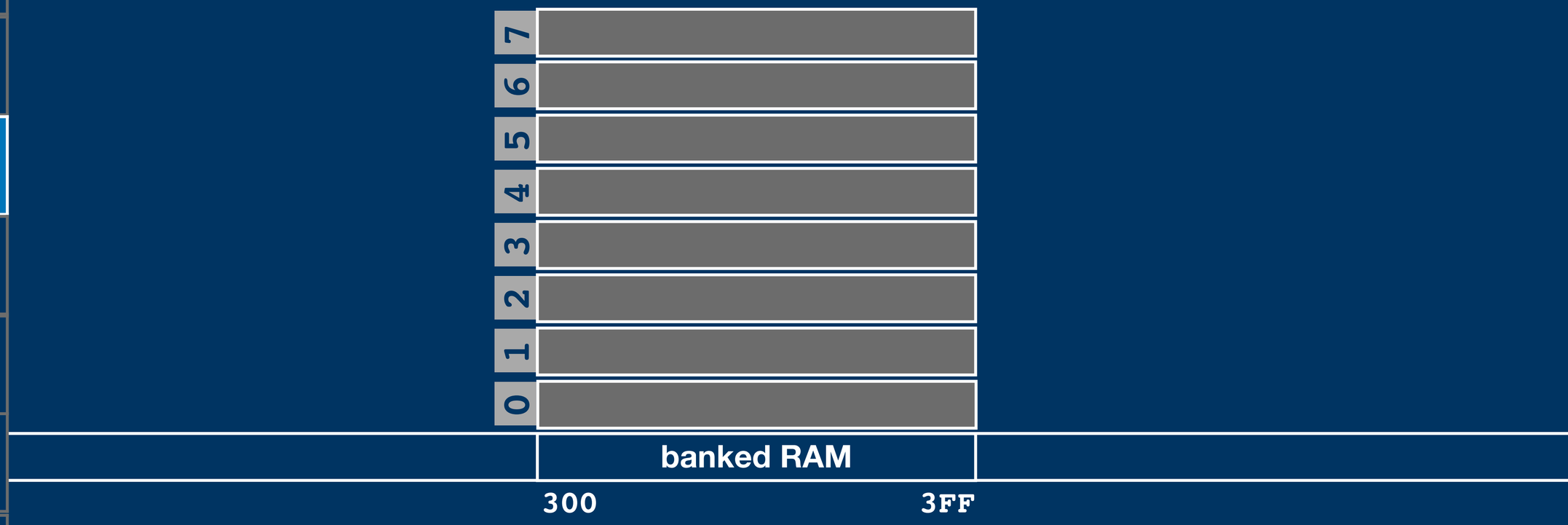
Memory Map


32 × 1024 W
Registers
8 × 256 W
fixed RAM
banked ROM
fixed ROM
000
300
400
800
FFF
banked
RAM

Switch RAM Bank

```
ld eb, a
```

A	0005
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

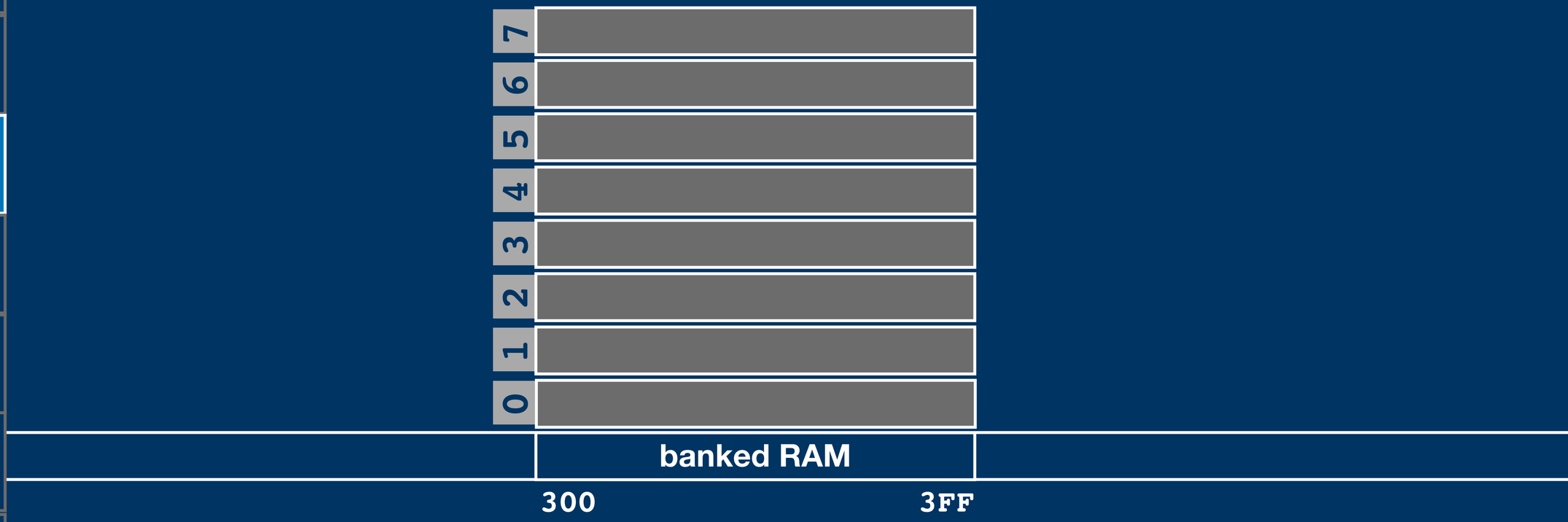


Switch RAM Bank

ld eb, a

Store	ld [k], a	TS k
	5800+k	2
	M[k] ← A	

A	0005
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

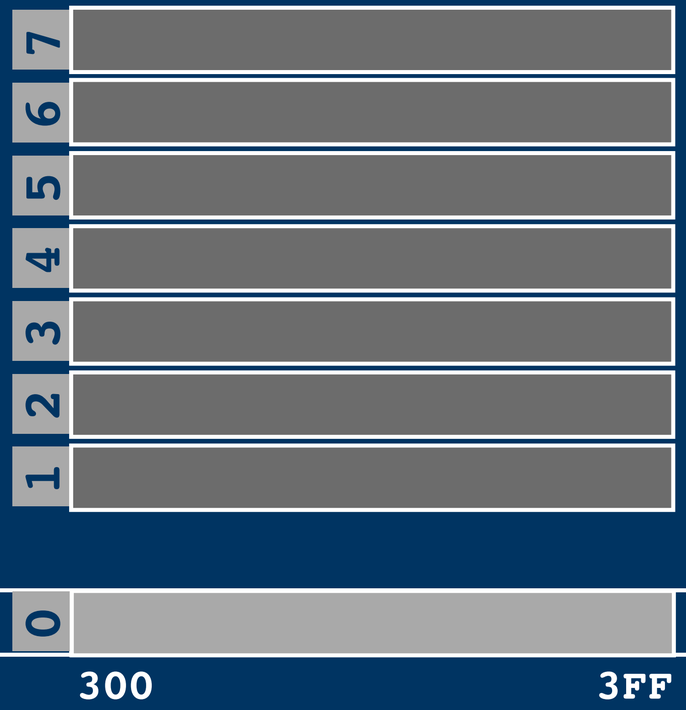


Switch RAM Bank

```
ld [3], a
```

Store	ld [k], a	TS k
	5800+k	2
	M[k] ← A	

A	0005	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

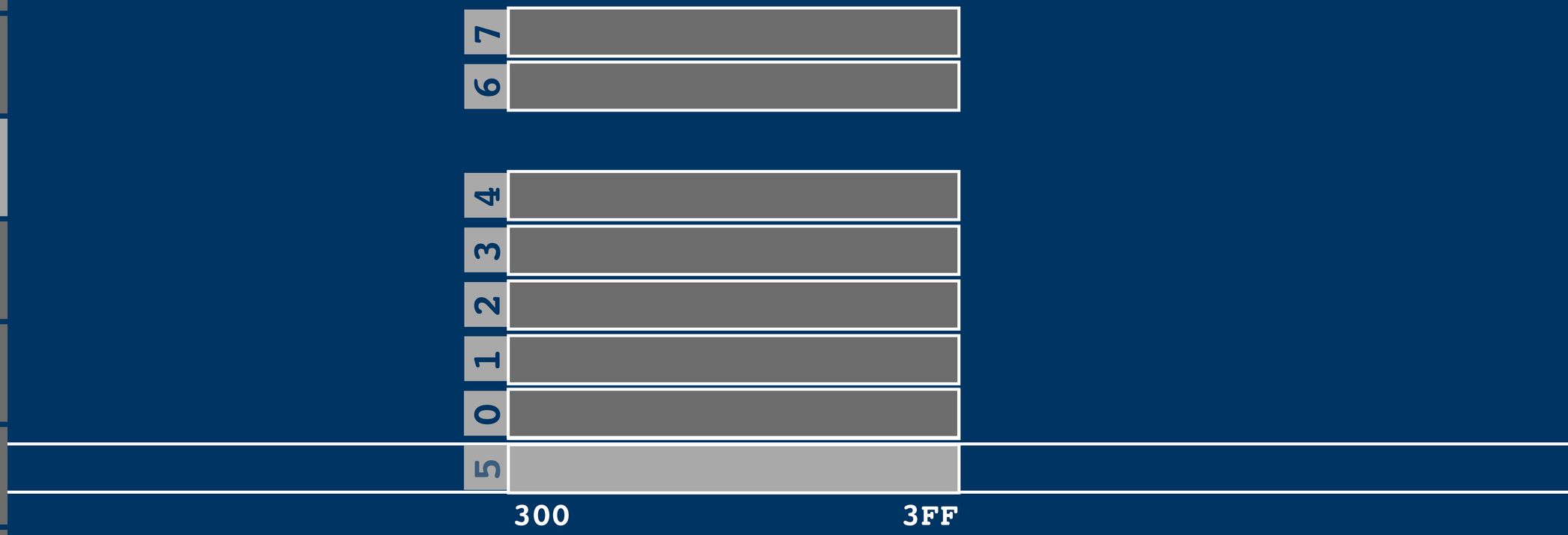


Switch RAM Bank

```
ld [3], a
```

Store	ld [k], a	TS k
	5800+k	2
	M[k] ← A	

A	0005	000
B	0F00	001
LR	05AA	002
EB	0005	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007



Switch ROM Bank

```
ld fb, a
```

A	0005	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	041F	005
BB	0000	006
0	0000	007

Store		
	ld [k], a	TS k
	5800+k	2
	M[k] ← A	

Switch ROM Bank

```
ld [4], a
```

Store	ld [k], a	TS k
	5800+k	2
	M[k] ← A	

A	0005	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	041F	005
BB	0000	006
0	0000	007

Switch ROM Bank

ld [4], a

Store	ld [k], a	ldx k
	5800+k	2
	M[k] ← A	

A	0005	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	041F	005
BB	0000	006
0	0000	007

00

5804	41F	ld fb, a
04E5	420	call $4e5

PC →

Switch ROM Bank

ld [4], a

Store	ld [k], a	ld k	
	5800+k	2	
	M[k] ← A		

A	0005	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0005	004
PC	041F	005
BB	0000	006
0	0000	007

00

5804	41F	ld fb, a
04E5	420	call $4e5

PC →

Switch ROM Bank

```
ld [4], a
```

Store	ld [k], a	ld k
	5800+k	2
	M[k] ← A	

A	0005	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0005	004
PC	041F	005
BB	0000	006
0	0000	007

05

373D	E	
3FE0	41F	ld a, [$fe0]
2BFF	420	inc [$3ff]
0000	42	

PC →

Call Far

callf

Call Far	callf	DTCF
	xchg ab, [4]	3
	A ↔ FB B ↔ PC	

A	0000	000
B	0000	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

Call Far

callf

A	0000	000
B	0000	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

Call Far

callf	DTCF
xchg ab, [4] (alias)	3
A ↔ FB B ↔ PC	

Call Far

callf

Call Far	callf	DTCF
	xchg ab, [4] (alias)	3
	A ↔ FB B ↔ PC	

A	4000	000
B	0400	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

PC →

00	3451	41F	ld ab, [$450]
00	5205	420	callf
00	3103	421	ld a, [$103]

04	5880	400	ld [$80], a
04	0002	401	retf

44D	0CDC	
44E	3193	
44F	4584	
450	4000	
451	0400	
452	37B5	
453	52A3	
454	24D1	

Call Far

callf

Call Far	callf	DTCF
	xchg ab, [4] (alias)	3
	A ↔ FB B ↔ PC	

A	0000	000
B	0421	001
LR	05AA	002
EB	0000	003
FB	4000	004
PC	0400	005
BB	0000	006
0	0000	007

00	3451	41F	ld ab, [$450]
00	5205	420	callf
00	3103	421	ld a, [$103]

PC →

04	5880	400	ld [$80], a
04	0002	401	retf

44D	0CDC	
44E	3193	
44F	4584	
450	4000	
451	0400	
452	37B5	
453	52A3	
454	24D1	

Return Far

retf

Return Far	retf	DTCF
	xchg ab, [4] (alias)	3
	A ↔ FB B ↔ PC	

A	0000	000
B	0421	001
LR	05AA	002
EB	0000	003
FB	4000	004
PC	0400	005
BB	0000	006
0	0000	007

00	3451	41F	ld ab, [$450]
00	5205	420	callf
00	3103	421	ld a, [$103]

PC →

04	5880	400	ld [$80], a
04	0002	401	retf

Return Far

retf

Return Far	retf	DTCF
	xchg ab, [4] (alias)	3
	A ↔ FB B ↔ PC	

A	4000	000
B	0400	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

	3451	41F		ld ab, [$450]
	5205	420		callf
PC →	3103	421		ld a, [$103]
	5880	400		ld [$80], a
	0002	401		retf

Bank Registers

Bank Registers

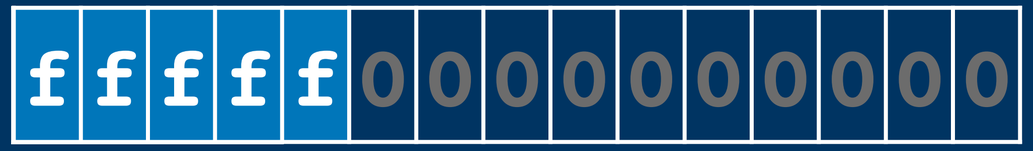
FB 0000

EB 0000

Bank Registers

5 bits
32 banks

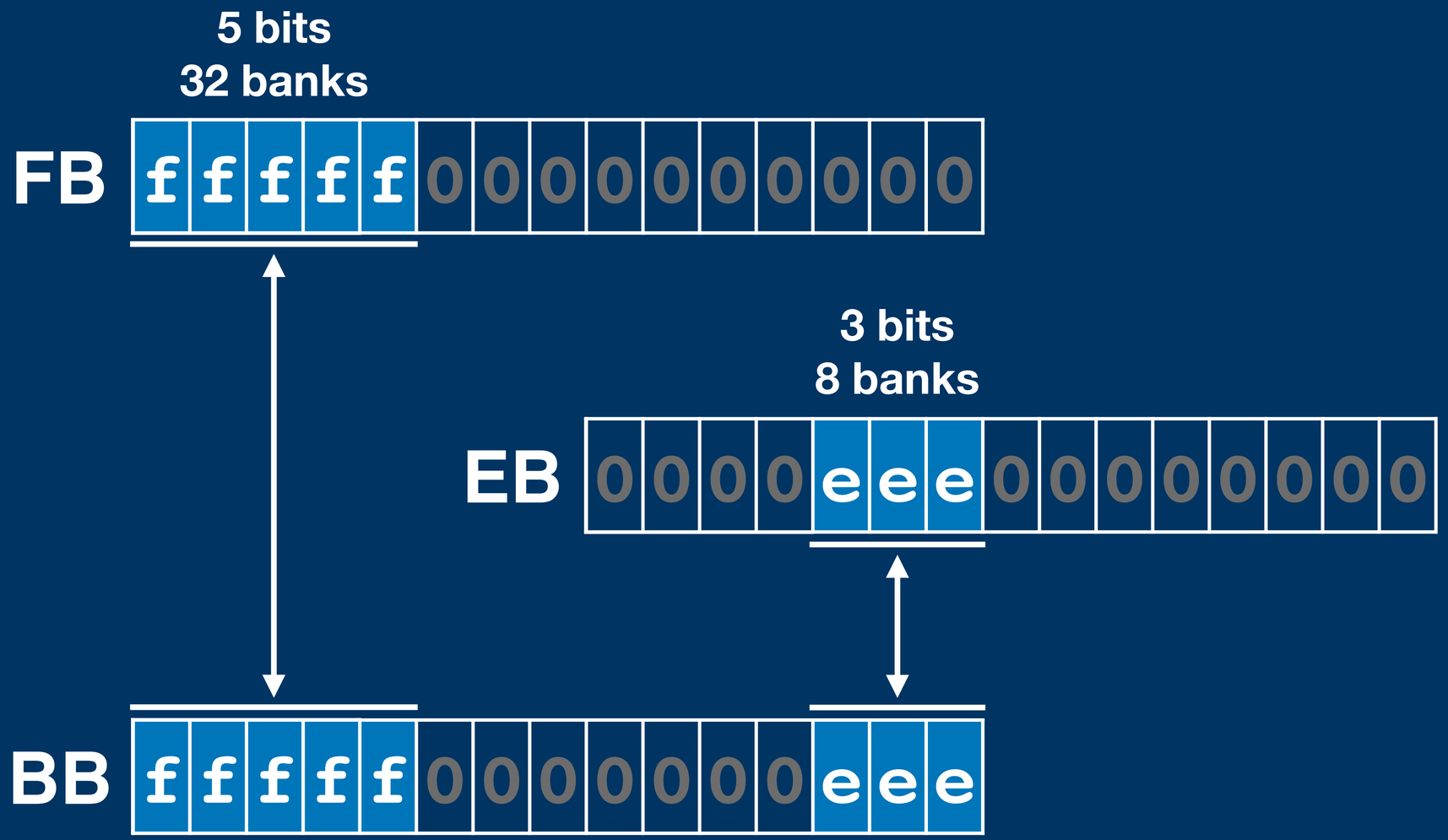
FB



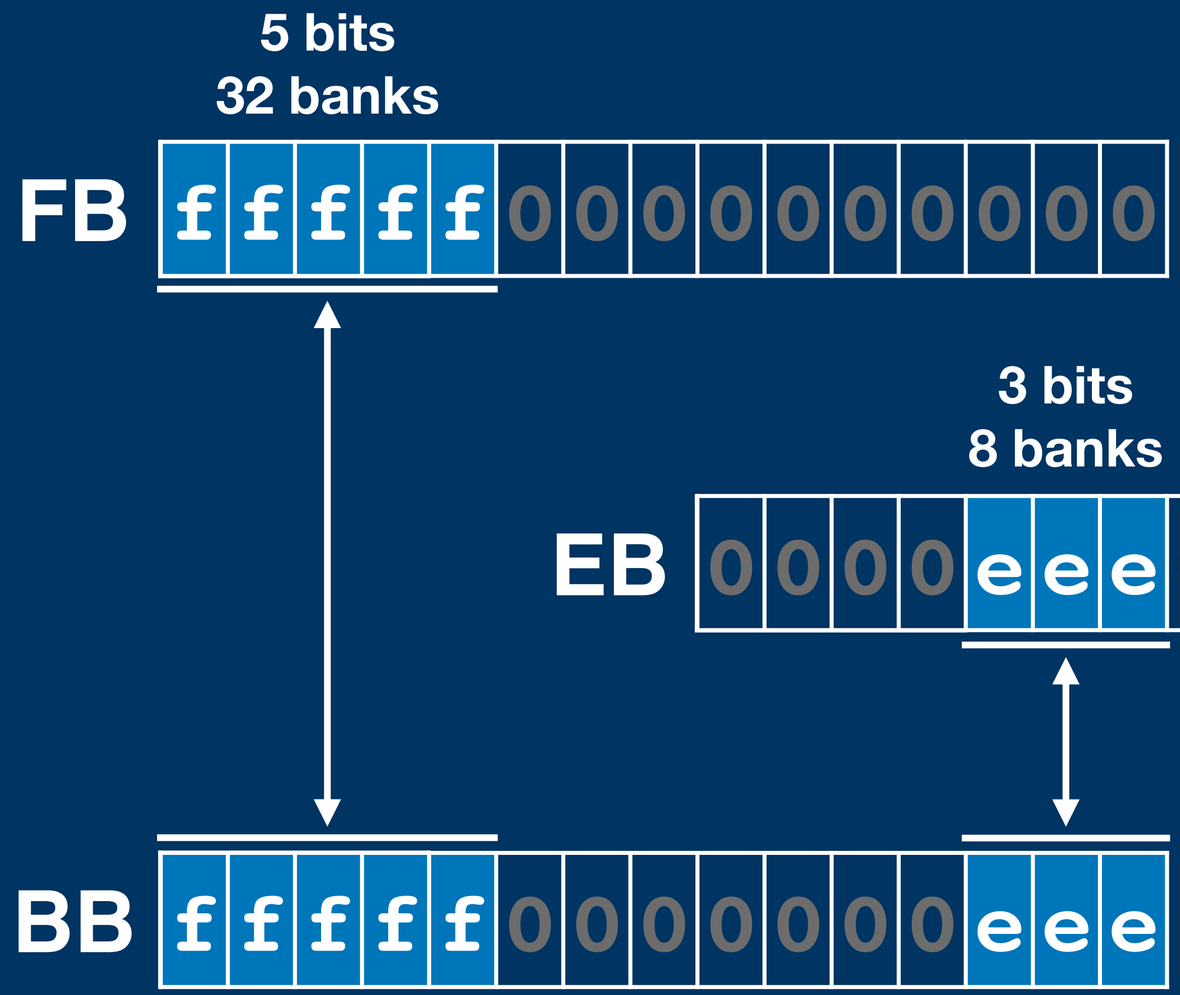
3 bits
8 banks

EB

Bank Registers



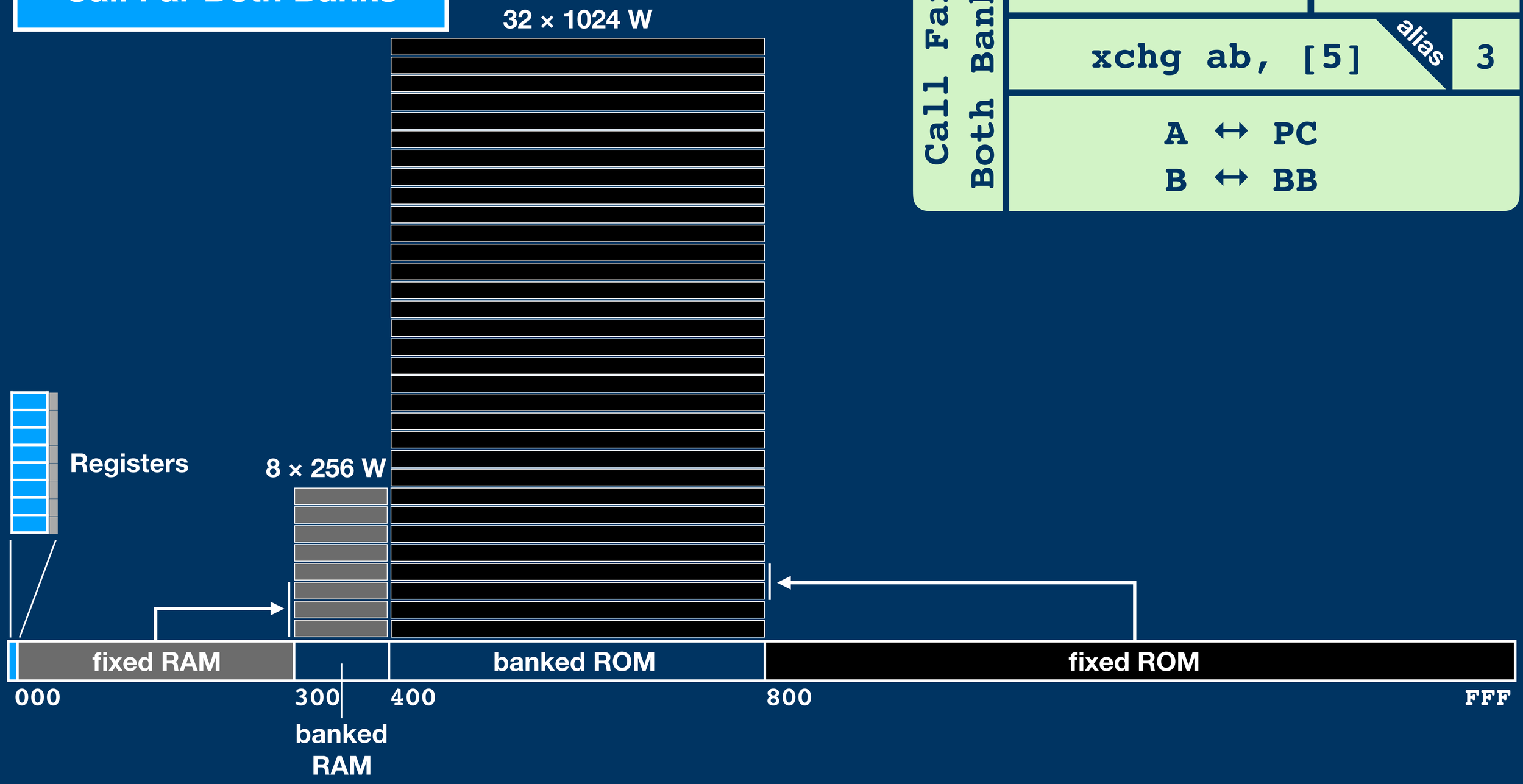
Bank Registers



Call Far Both Banks	callfbb	DTCB
	xchg ab, [5] (alias)	3
	A ↔ PC B ↔ BB	

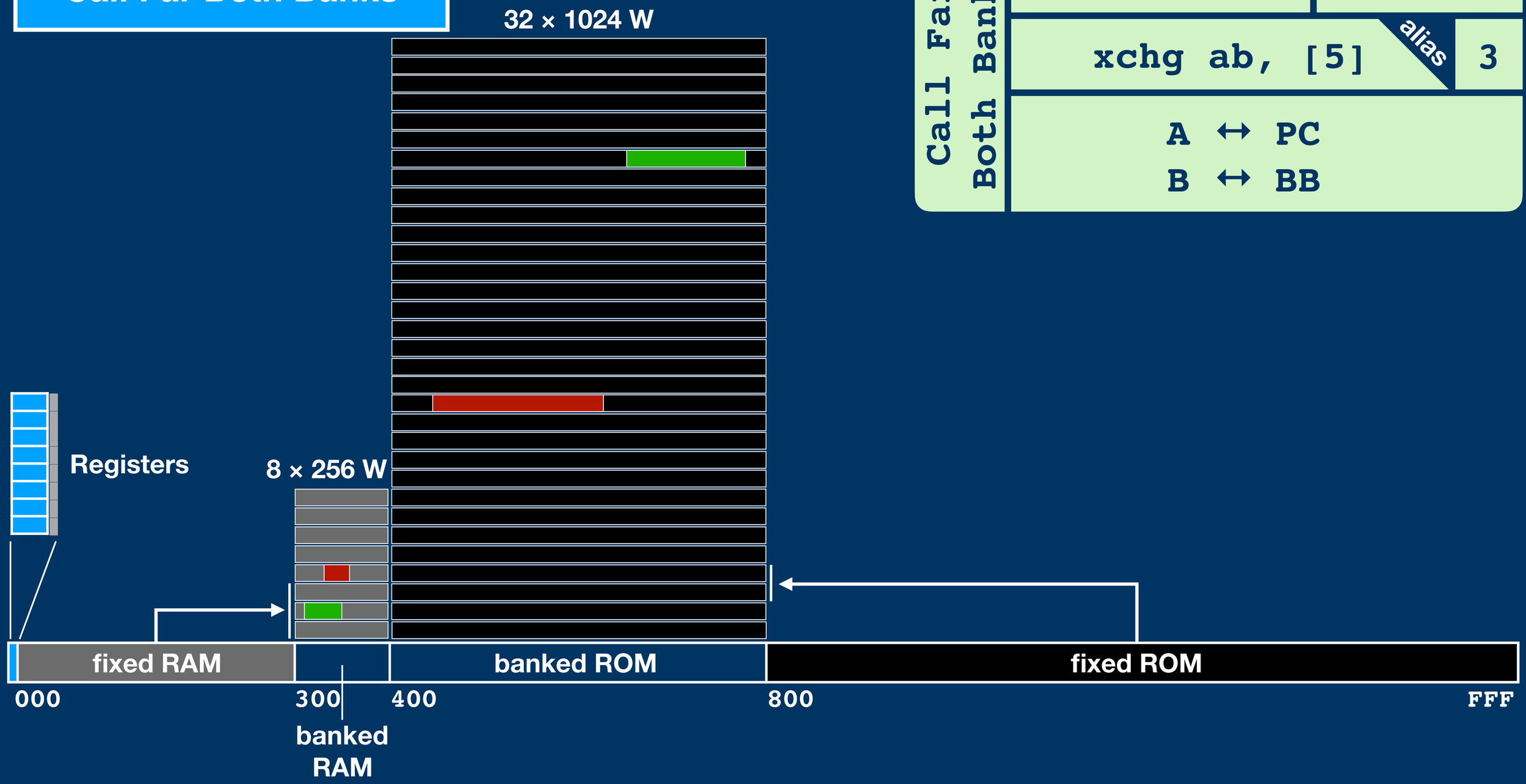
Call Far Both Banks

Call Far Both Banks	callfbb	DTCB
	xchg ab, [5] (alias)	3
	A ↔ PC B ↔ BB	

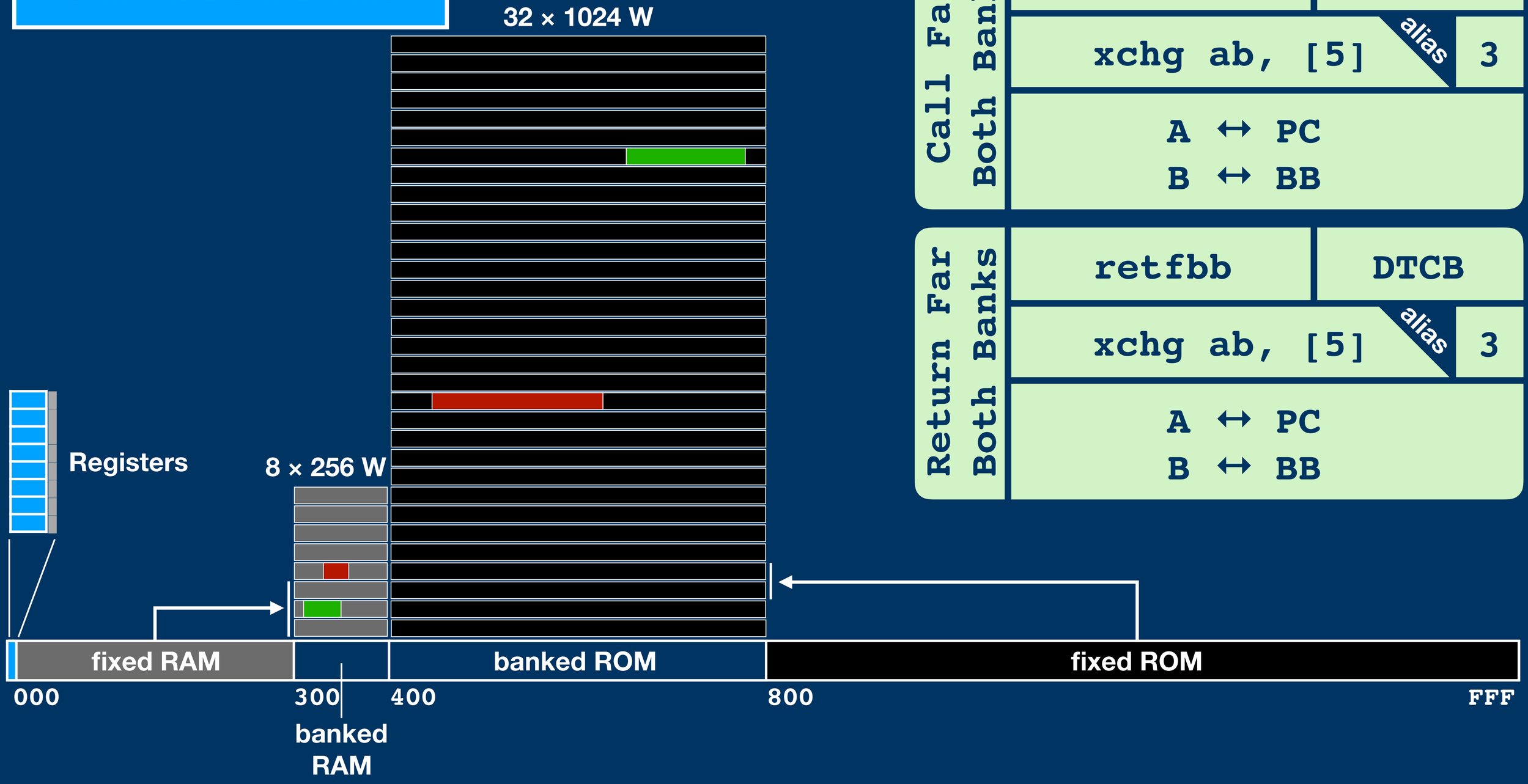


Call Far Both Banks

Call Far Both Banks	callfbb	DTCB
	xchg ab, [5] (alias)	3
	A ↔ PC B ↔ BB	



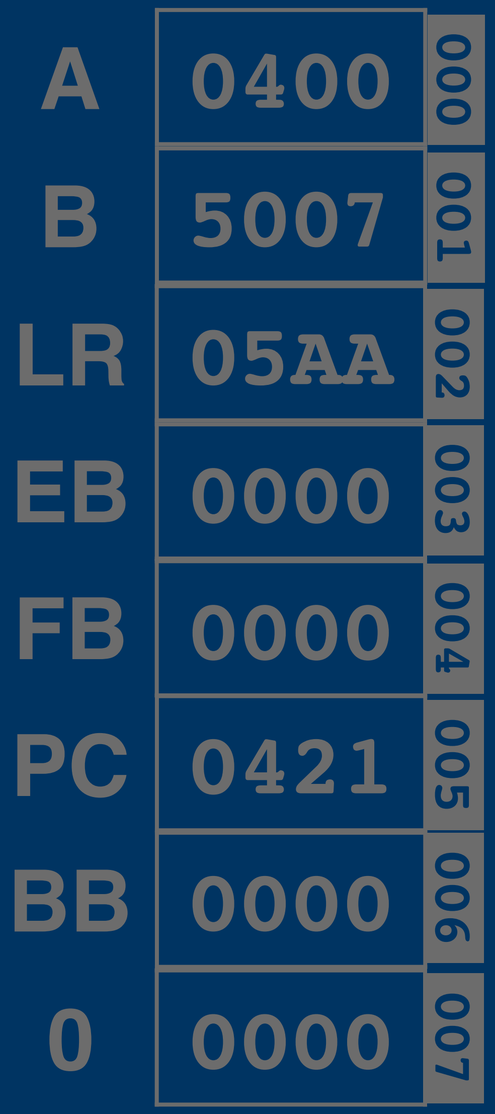
Call Far Both Banks



Call Far Both Banks		
	callfbb	DTCB
	xchg ab, [5] (alias)	3
	A ↔ PC B ↔ BB	

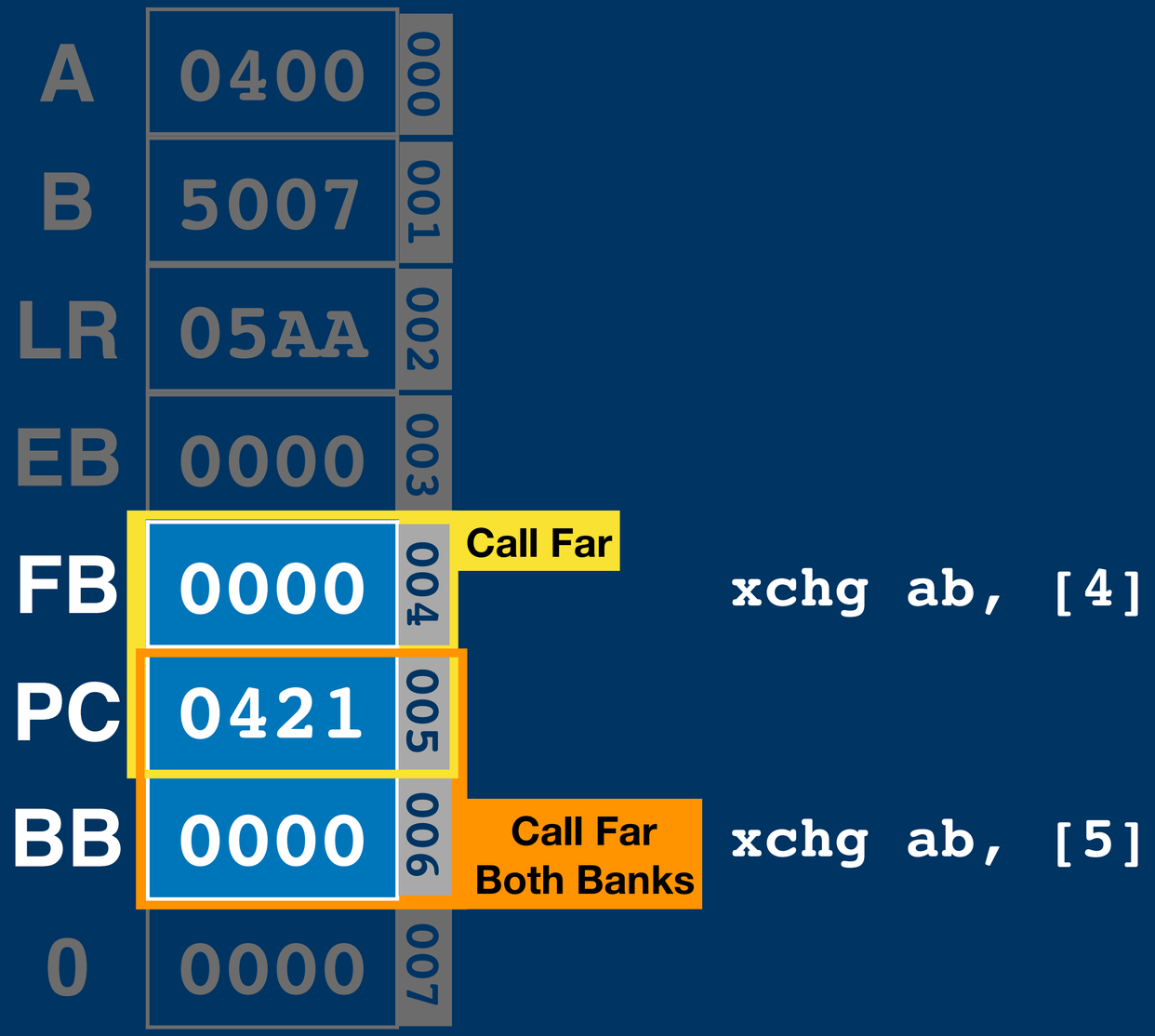
Return Far Both Banks		
	retfbb	DTCB
	xchg ab, [5] (alias)	3
	A ↔ PC B ↔ BB	

Register Ordering

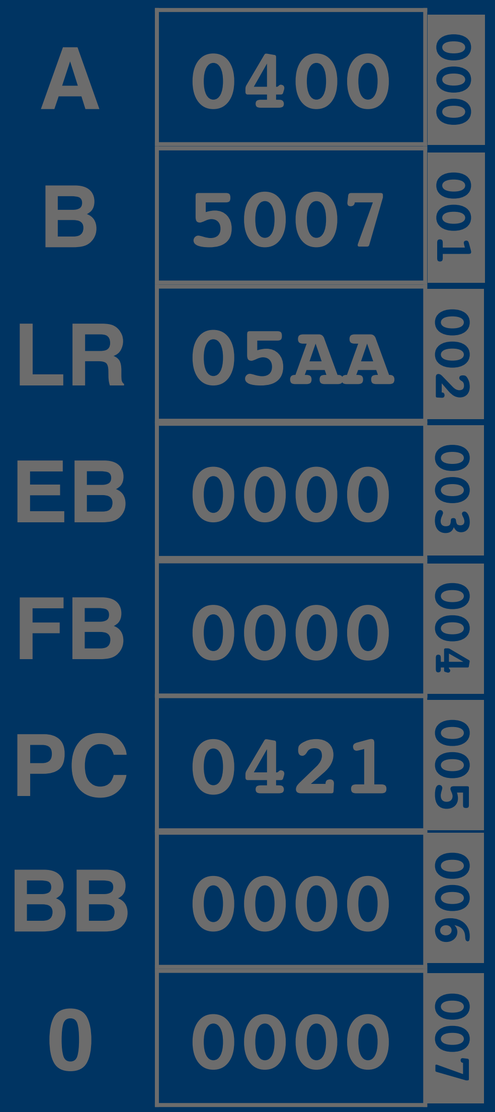


Register Ordering

Register Ordering



Registers


A
0400
000
B
5007
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007

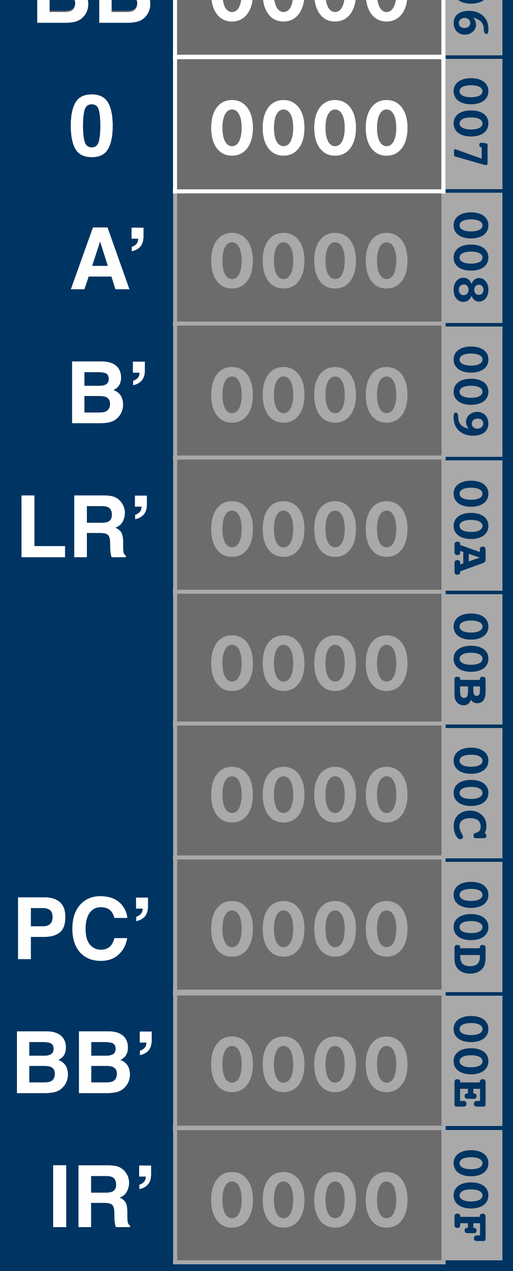
Registers


A
0007
000
B
0F00
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007

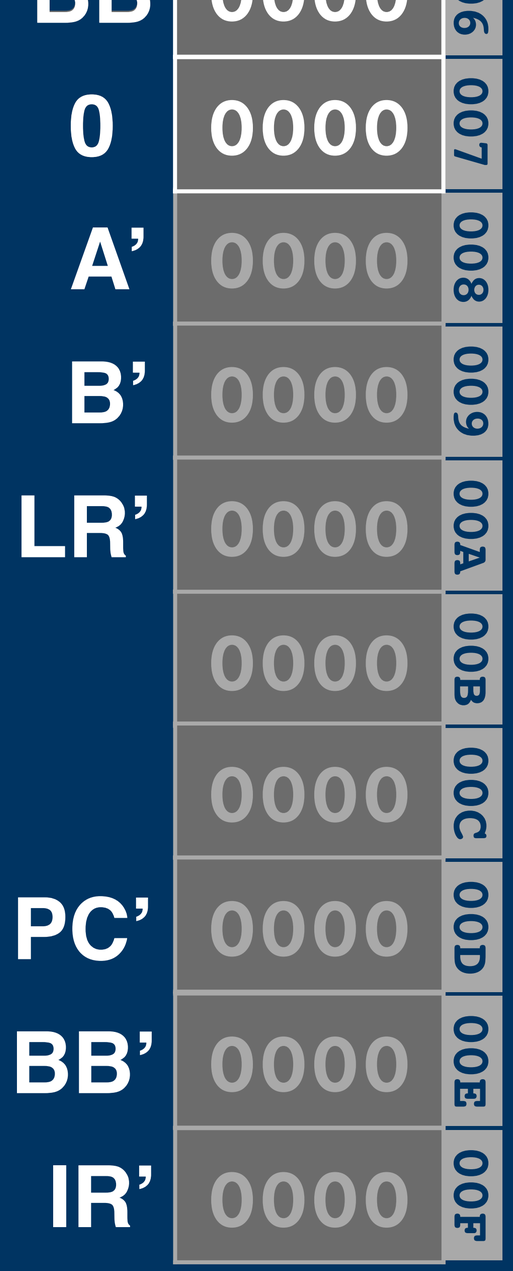
Registers


A
0007
000
B
0F00
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007

Registers


BB
0000
006
0
0000
007
A’
0000
008
B’
0000
009
LR’
0000
00A
0000
00B
0000
00C
PC’
0000
00D
BB’
0000
00E
IR’
0000
00F

Editing

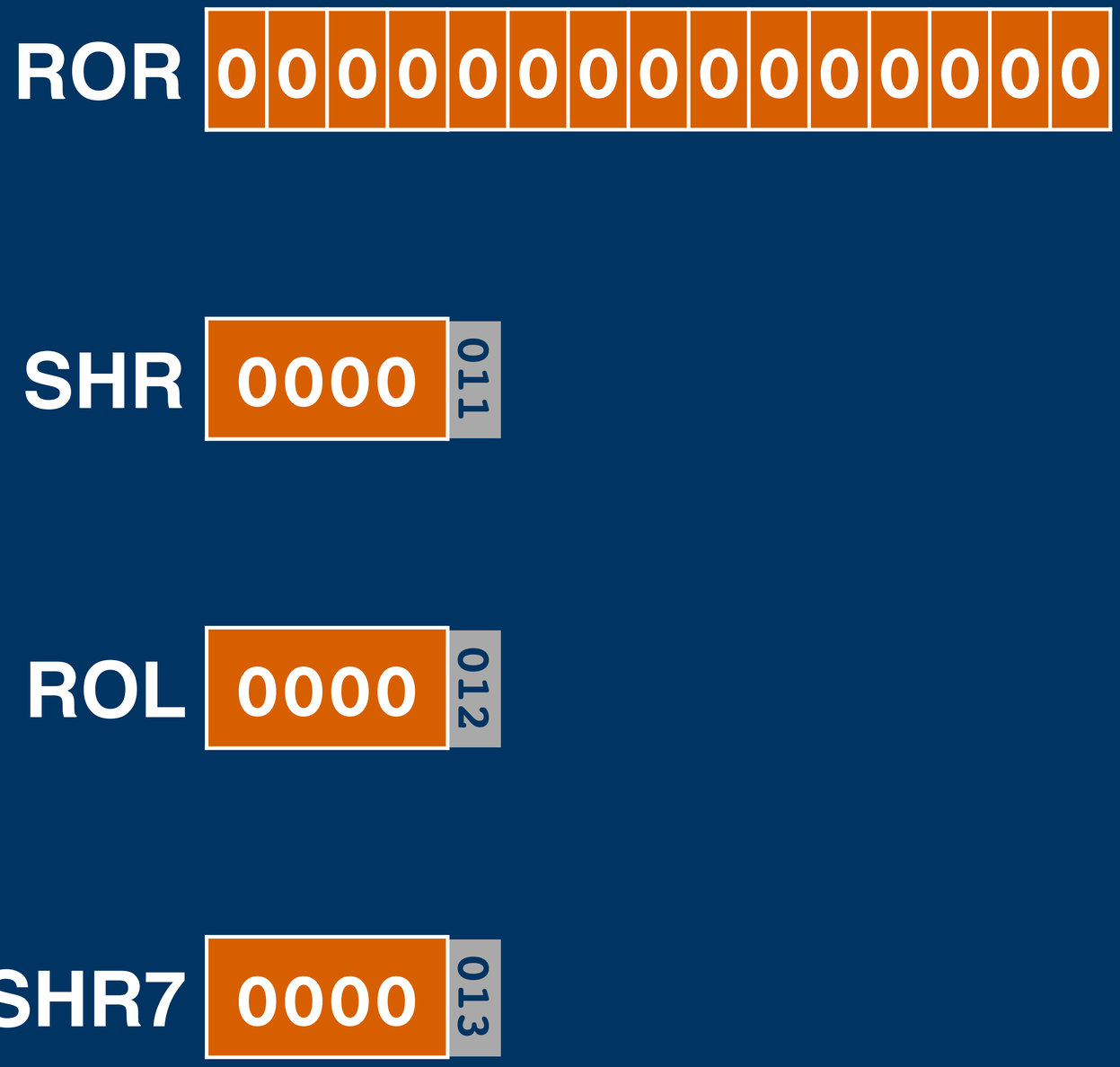

BB
0000
006
0
0000
007
A’
0000
008
B’
0000
009
LR’
0000
00A
0000
00B
0000
00C
PC’
0000
00D
BB’
0000
00E
IR’
0000
00F

Editing

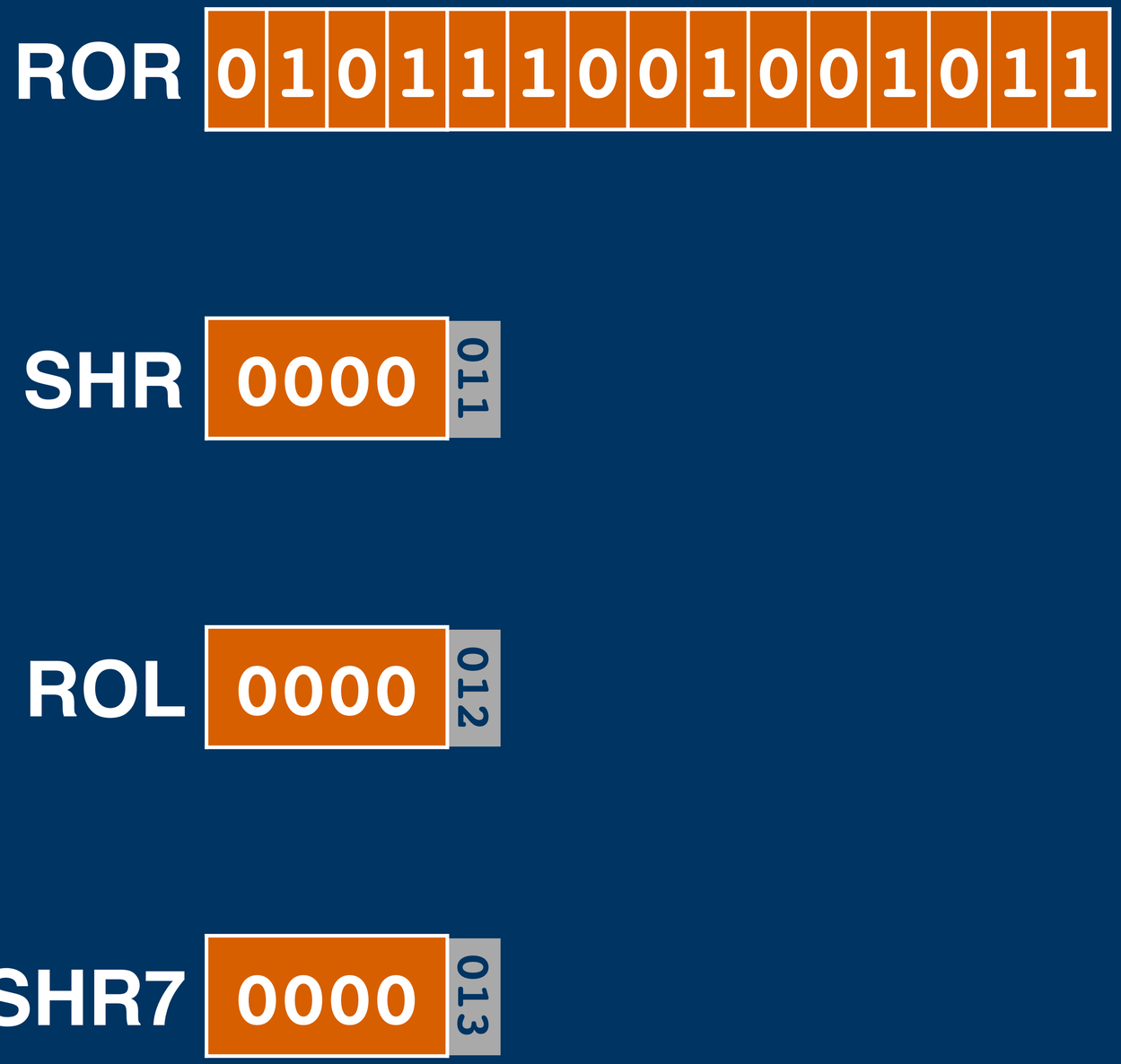

BB
0000
00E
IR’
0000
00F
ROR
0000
010
SHR
0000
011
ROL
0000
012
SHR7
0000
013

Editing

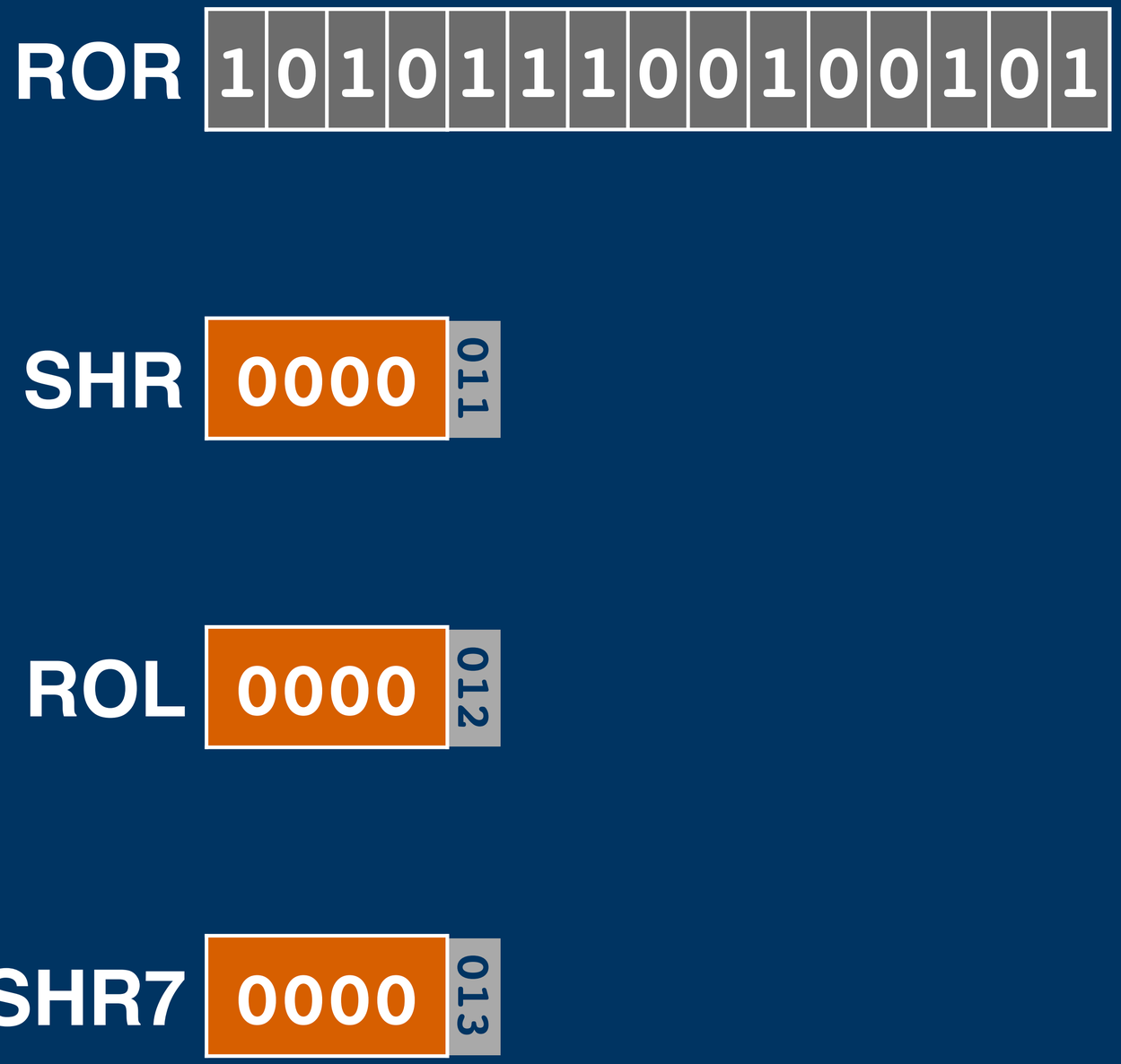
Editing


ROR
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SHR
0000
011
ROL
0000
012
SHR7
0000
013

Editing



Editing



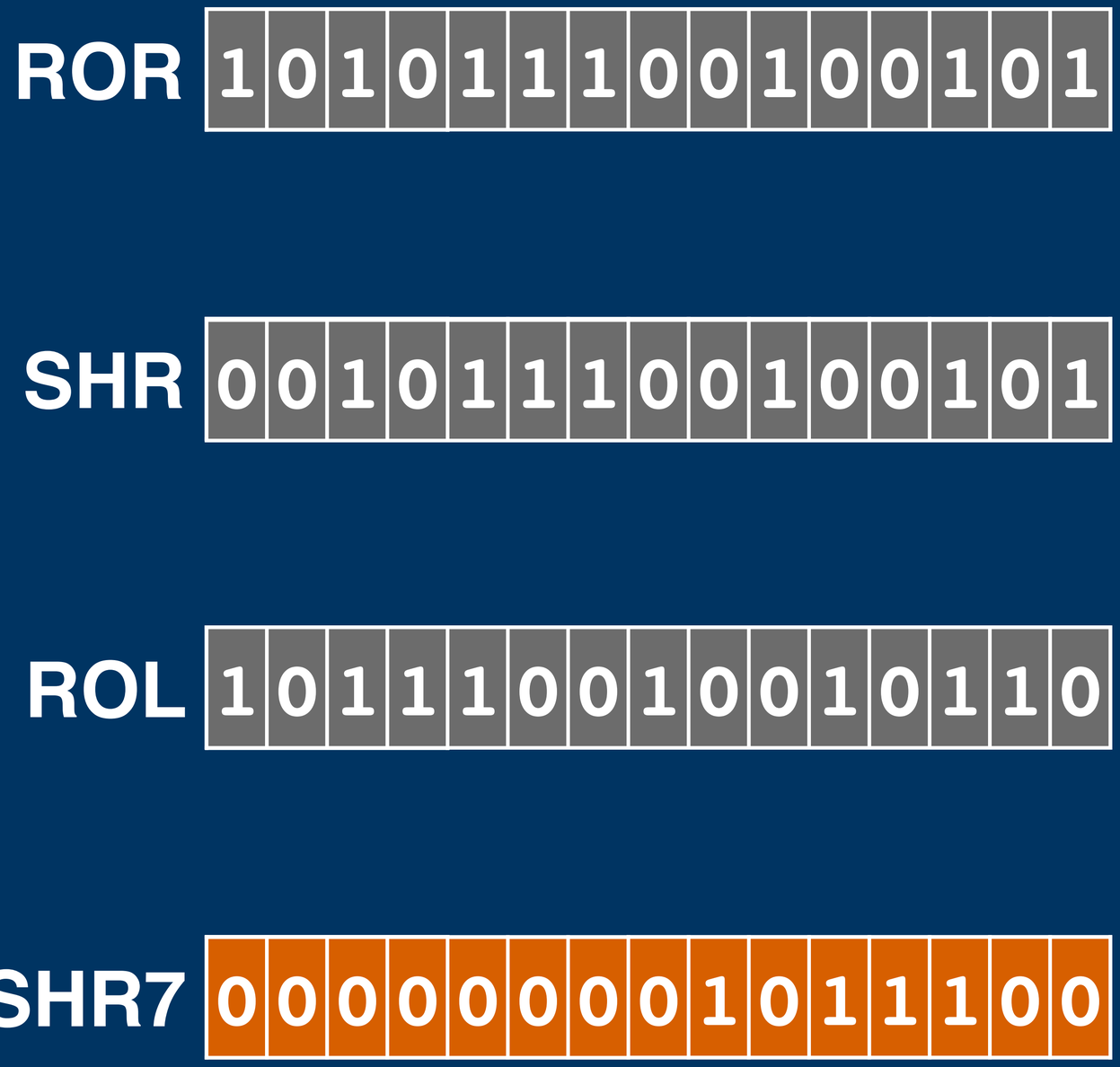
Editing


ROR
1 0 1 0 1 1 1 0 0 1 0 0 1 0 1
SHR
0000
011
ROL
1 0 1 1 1 0 0 1 0 0 1 0 1 1 0
SHR7
0000
013

Editing

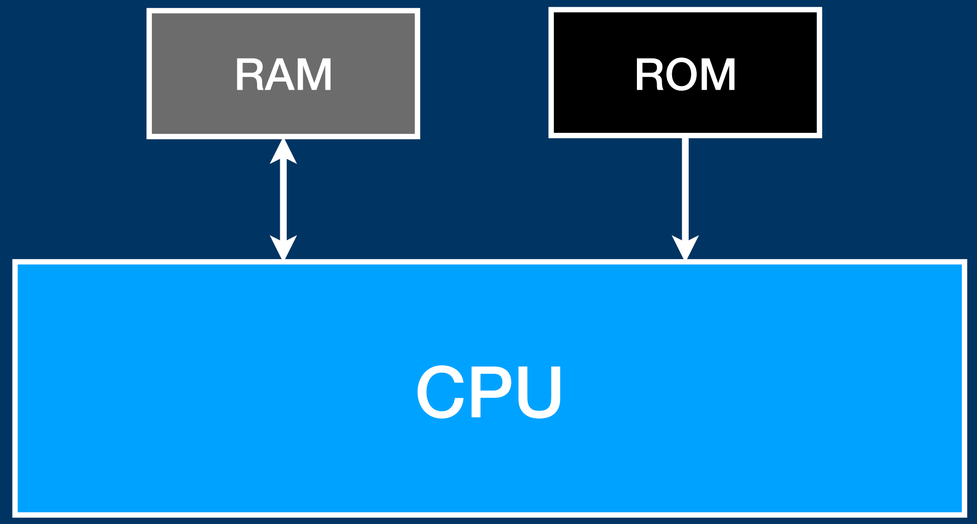

ROR
1 0 1 0 1 1 1 0 0 1 0 0 1 0 1
SHR
0 0 1 0 1 1 1 0 0 1 0 0 1 0 1
ROL
1 0 1 1 1 0 0 1 0 0 1 0 1 1 0
SHR7
0000
013

Editing

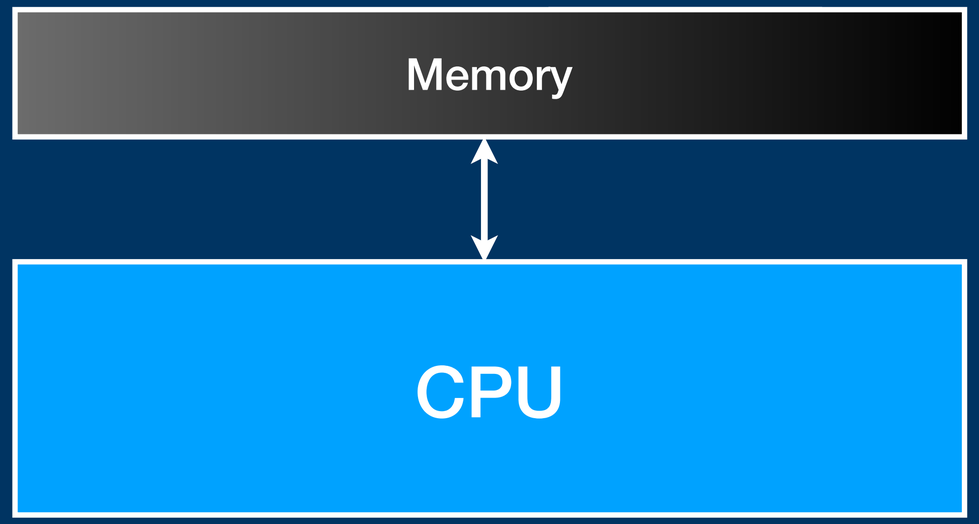

ROR 1 0 1 0 1 1 1 0 0 1 0 0 1 0 1
SHR 0 0 1 0 1 1 1 0 0 1 0 0 1 0 1
ROL 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0
SHR7 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0

I/O Channels

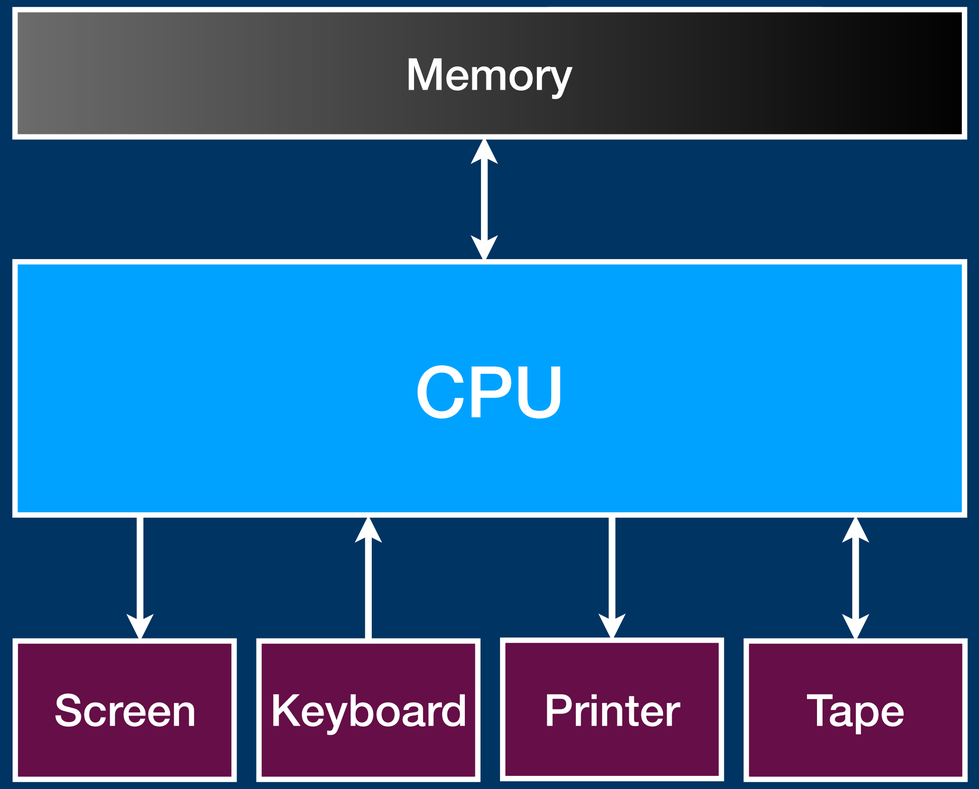
I/O Channels



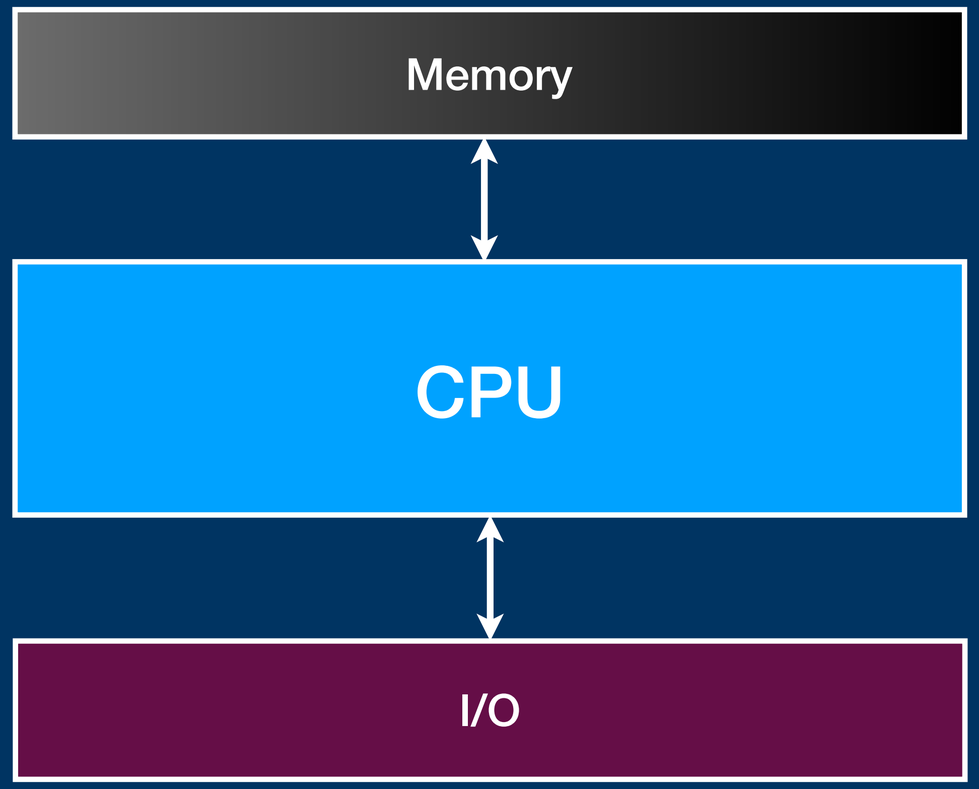
I/O Channels

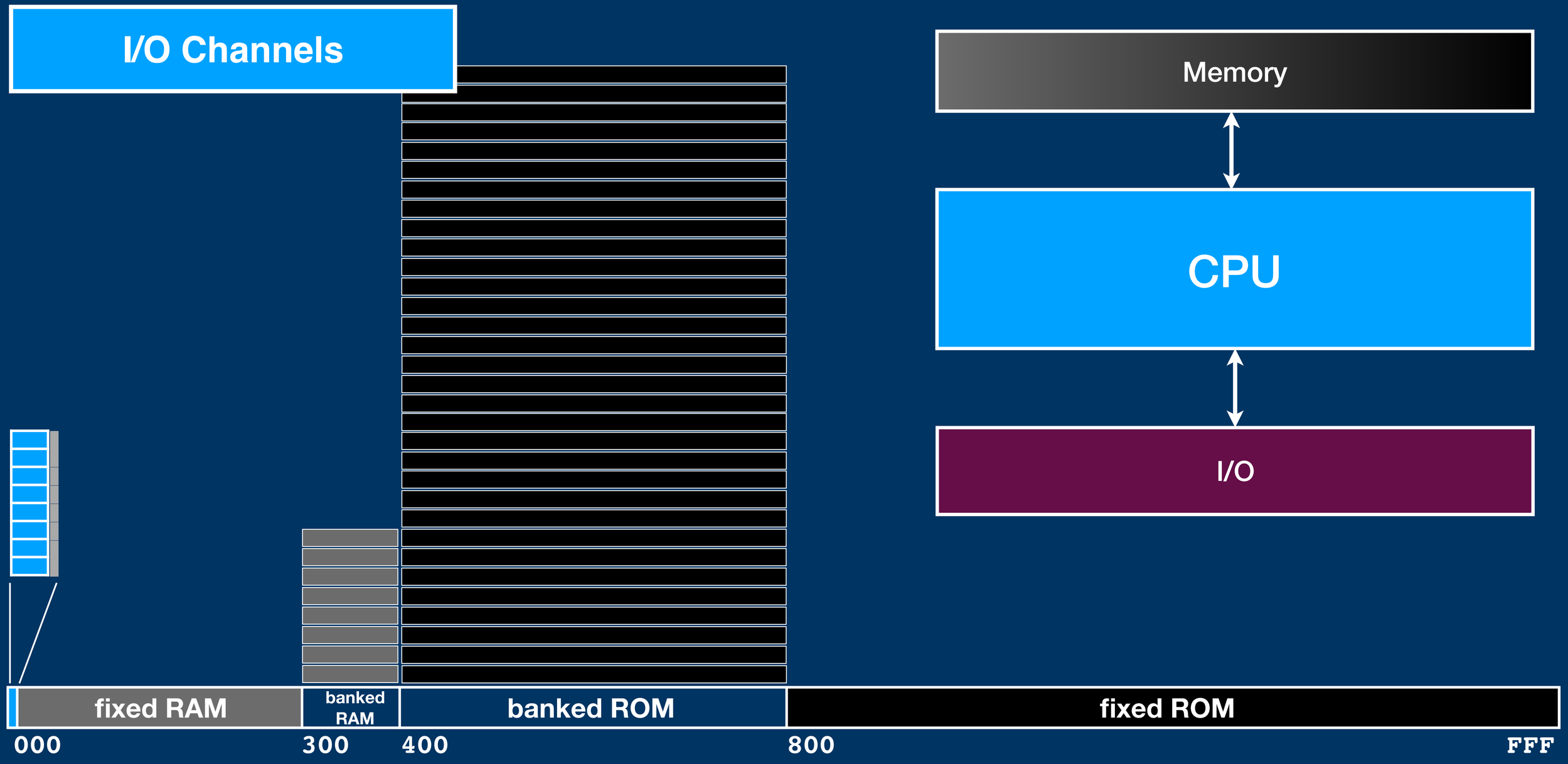


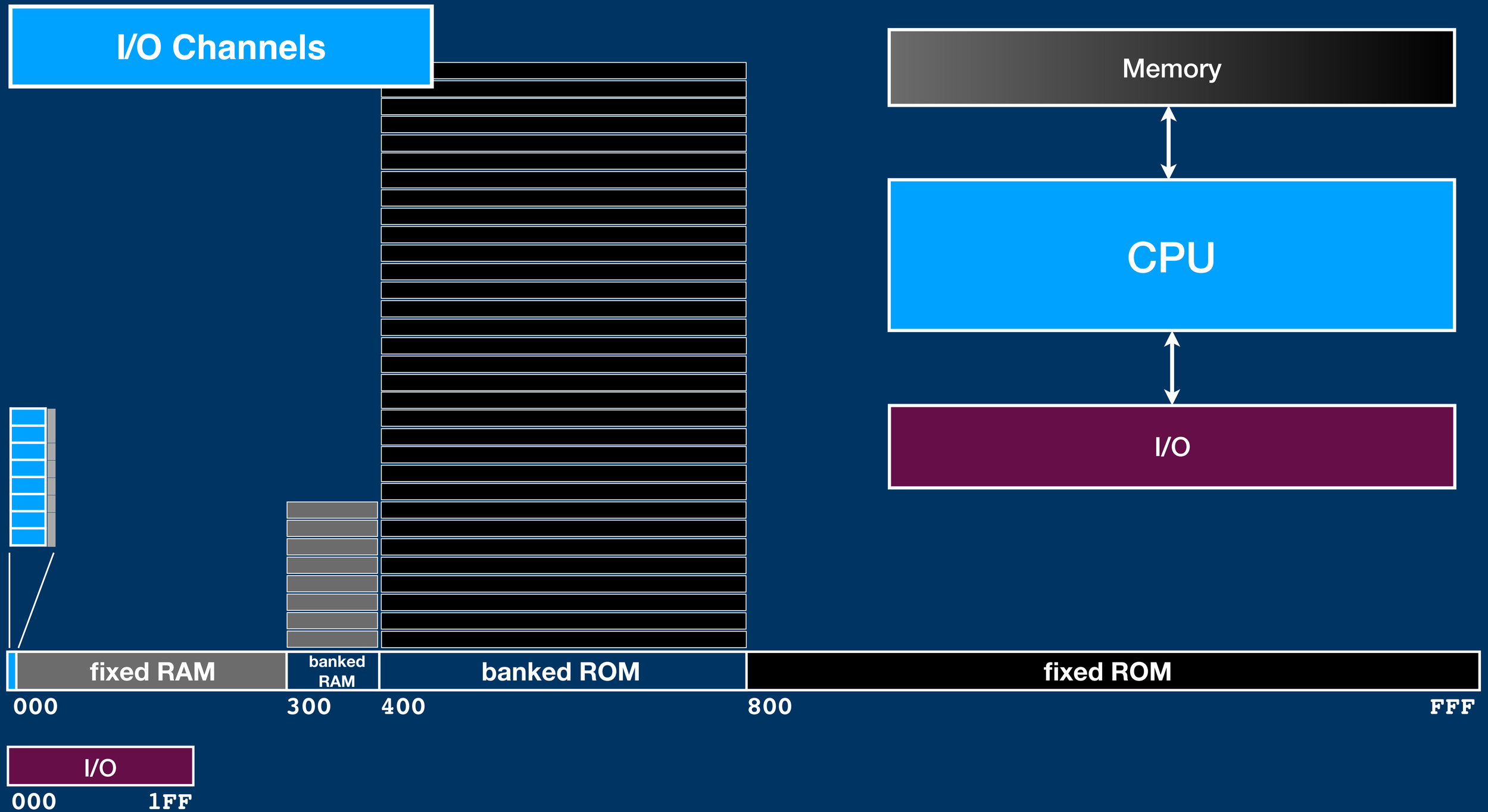
I/O Channels



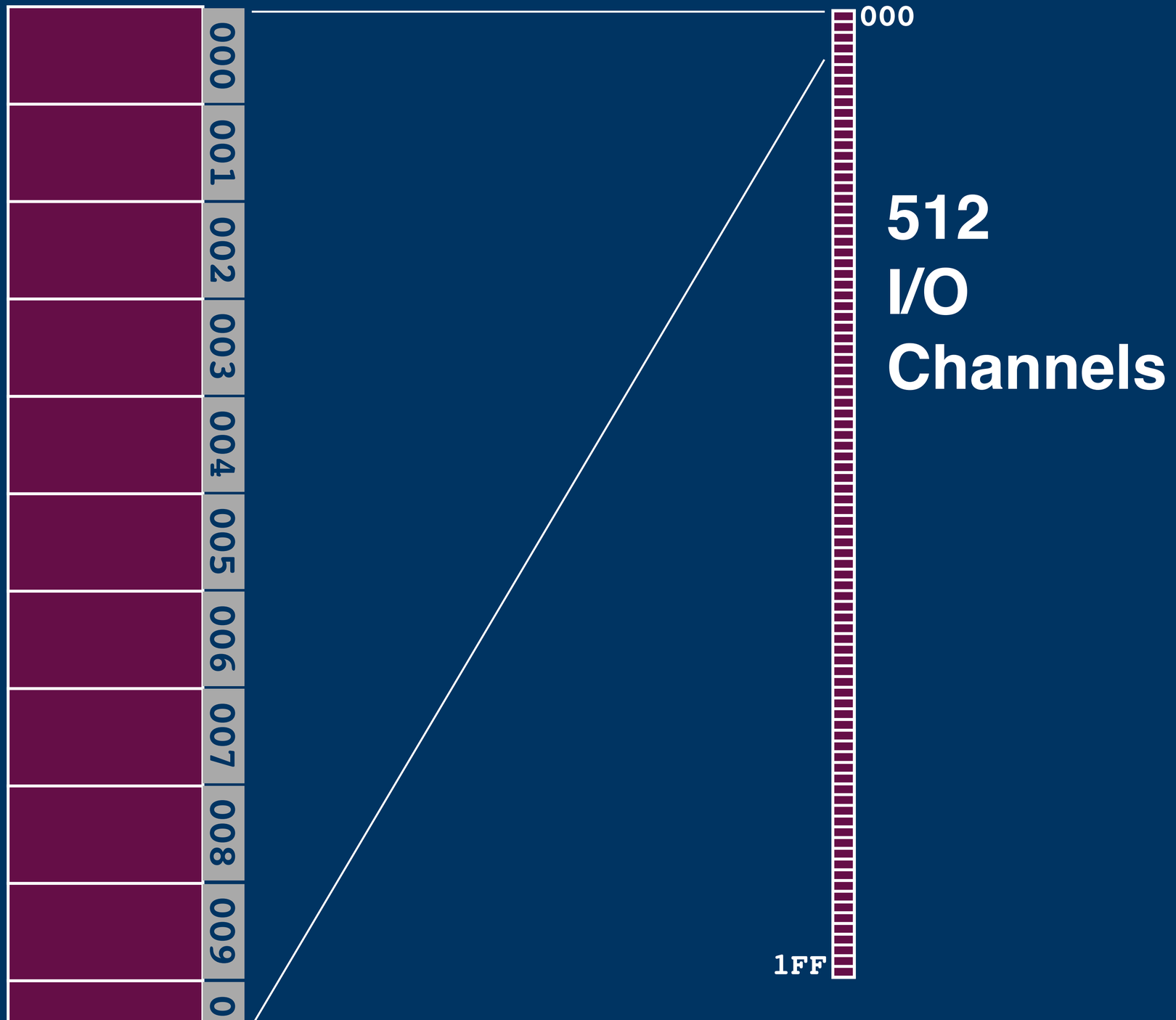
I/O Channels



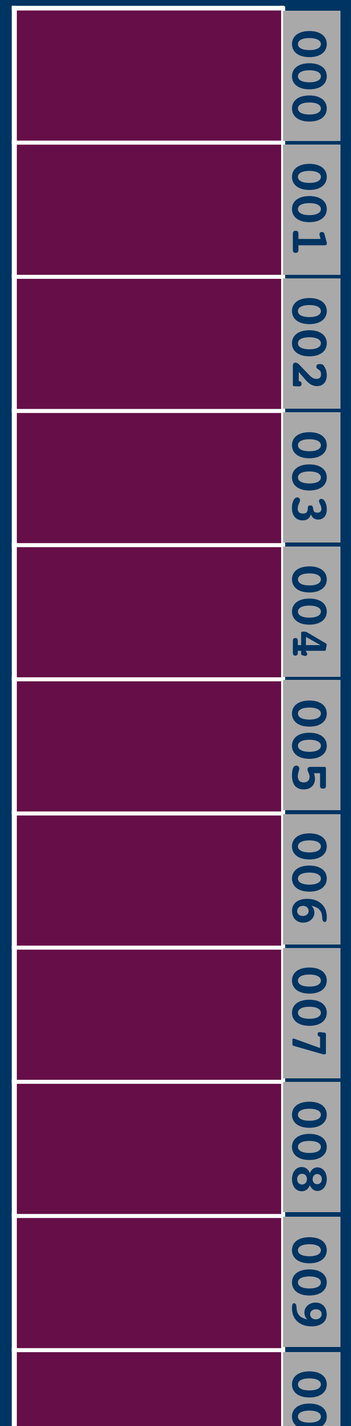

I/O Channels
Memory
CPU
I/O
fixed RAM
banked RAM
banked ROM
fixed ROM
000
300
400
800
FFF


I/O Channels
Memory
CPU
I/O
fixed RAM
banked RAM
banked ROM
fixed ROM
000
300
400
800
FFF
I/O
000
1FF


000
1FF


000
001
002
003
004
005
006
007
008
009
000
1FF
512
I/O
Channels

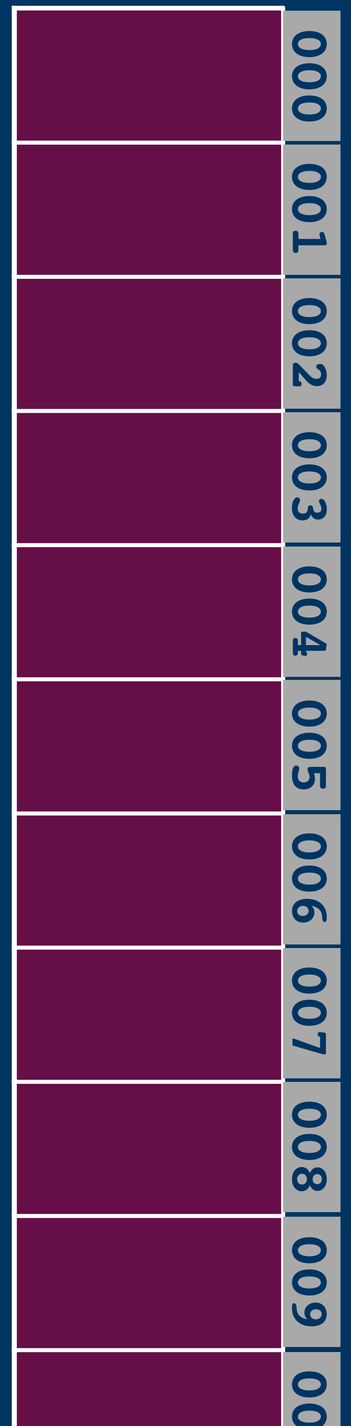
I/O Channels



I/O Channels

In	`in a, [kc]`	`READ kc`
	`0006, 0000+kc`	`3`
	`A ← IO[kc]`	

Out	`out [kc], a`	`WRITE kc`
	`0006, 0200+kc`	`3`
	`IO[kc] ← A`	

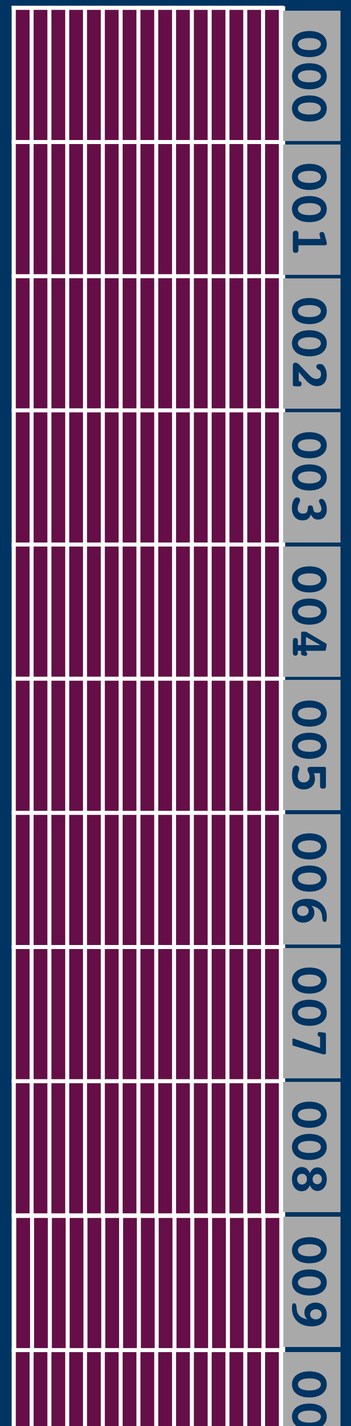


15 bit

I/O Channels

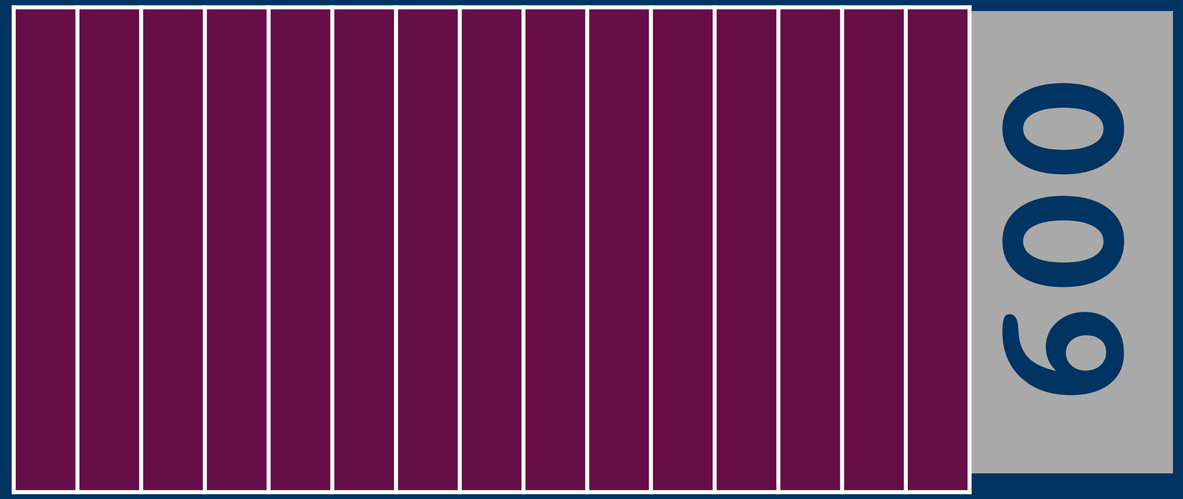
In		
	`in a, [kc]`	READ kc
	0006, 0000+kc	3
	A ← IO[kc]	

Out		
	`out [kc], a`	WRITE kc
	0006, 0200+kc	3
	IO[kc] ← A	



15 bit

I/O Channels



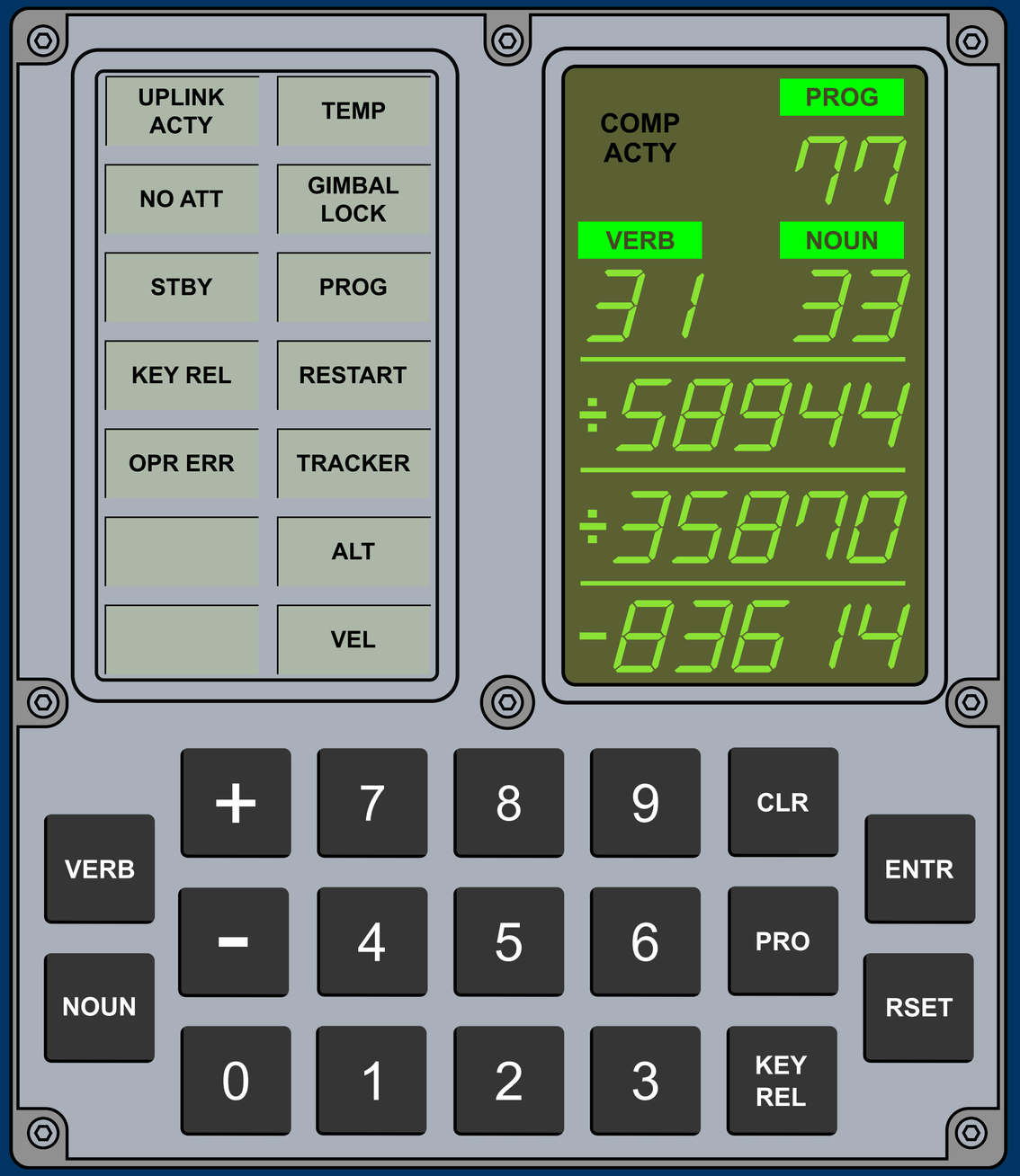
I/O Channels

009

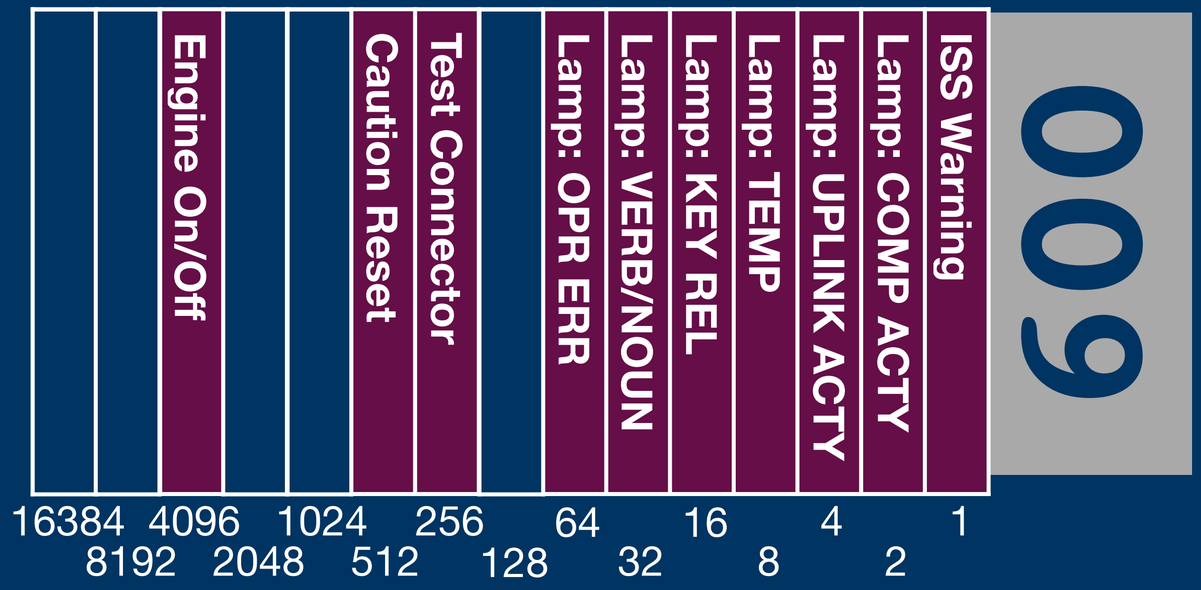
ISS Warning
Lamp: COMP ACTY
Lamp: UPLINK ACTY
Lamp: TEMP
Lamp: KEY REL
Lamp: VERB/NOUN
Lamp: OPR ERR
Test Connector
Caution Reset
Engine On/Off

I/O Channels

009
ISS Warning
Lamp: COMP ACTY
Lamp: UPLINK ACTY
Lamp: TEMP
Lamp: KEY REL
Lamp: VERB/NOUN
Lamp: OPR ERR
Test Connector
Caution Reset
Engine On/Off

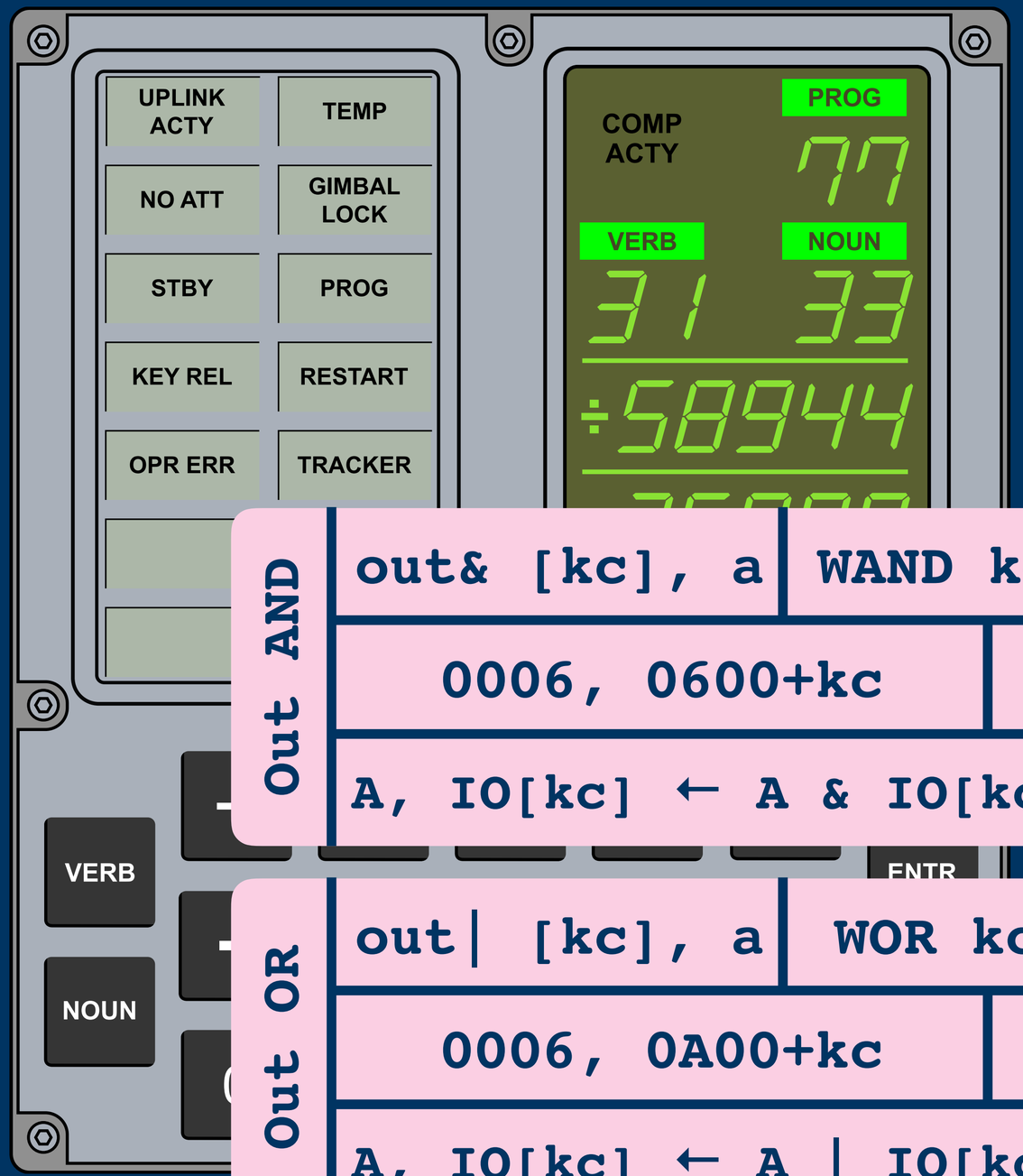


I/O Channels




```
ld a, #2
out| [9]
ld a, #~2
```

PC → `out& [9]`



Out AND

`out& [kc], a`	WAND kc
0006, 0600+kc	3
A, IO[kc] ← A & IO[kc]	

Out OR

`out| [kc], a`	WOR kc
0006, 0A00+kc	3
A, IO[kc] ← A | IO[kc]	

I/O Channels

Out AND		
out& [kc], a		WAND kc
0006, 0600+kc		3
A, IO[kc] ← A & IO[kc]		

Out OR		
out| [kc], a		WOR kc
0006, 0A00+kc		3
A, IO[kc] ← A | IO[kc]		

I/O Channels

In		
	`in a, [kc]`	`READ kc`
	`0006, 0000+kc`	`3`
	`A ← IO[kc]`	

Out		
	`out [kc], a`	`WRITE kc`
	`0006, 0200+kc`	`3`
	`IO[kc] ← A`	

Out AND		
	`out& [kc], a`	`WAND kc`
	`0006, 0600+kc`	`3`
	`A, IO[kc] ← A & IO[kc]`	

Out OR		
	`out| [kc], a`	`WOR kc`
	`0006, 0A00+kc`	`3`
	`A, IO[kc] ← A | IO[kc]`	

I/O Channels

In		
	`in a, [kc]`	`READ kc`
	`0006, 0000+kc`	`3`
	`A ← IO[kc]`	

Out		
	`out [kc], a`	`WRITE kc`
	`0006, 0200+kc`	`3`
	`IO[kc] ← A`	

In AND		
	`in& a, [kc]`	`RAND kc`
	`0006, 0400+kc`	`3`
	`A ← A & IO[kc]`	

Out AND		
	`out& [kc], a`	`WAND kc`
	`0006, 0600+kc`	`3`
	`A, IO[kc] ← A & IO[kc]`	

Out OR		
	`out| [kc], a`	`WOR kc`
	`0006, 0A00+kc`	`3`
	`A, IO[kc] ← A | IO[kc]`	

I/O Channels

In		
in a, [kc]	READ kc	
0006, 0000+kc	3	
A ← IO[kc]		

Out		
out [kc], a	WRITE kc	
0006, 0200+kc	3	
IO[kc] ← A		

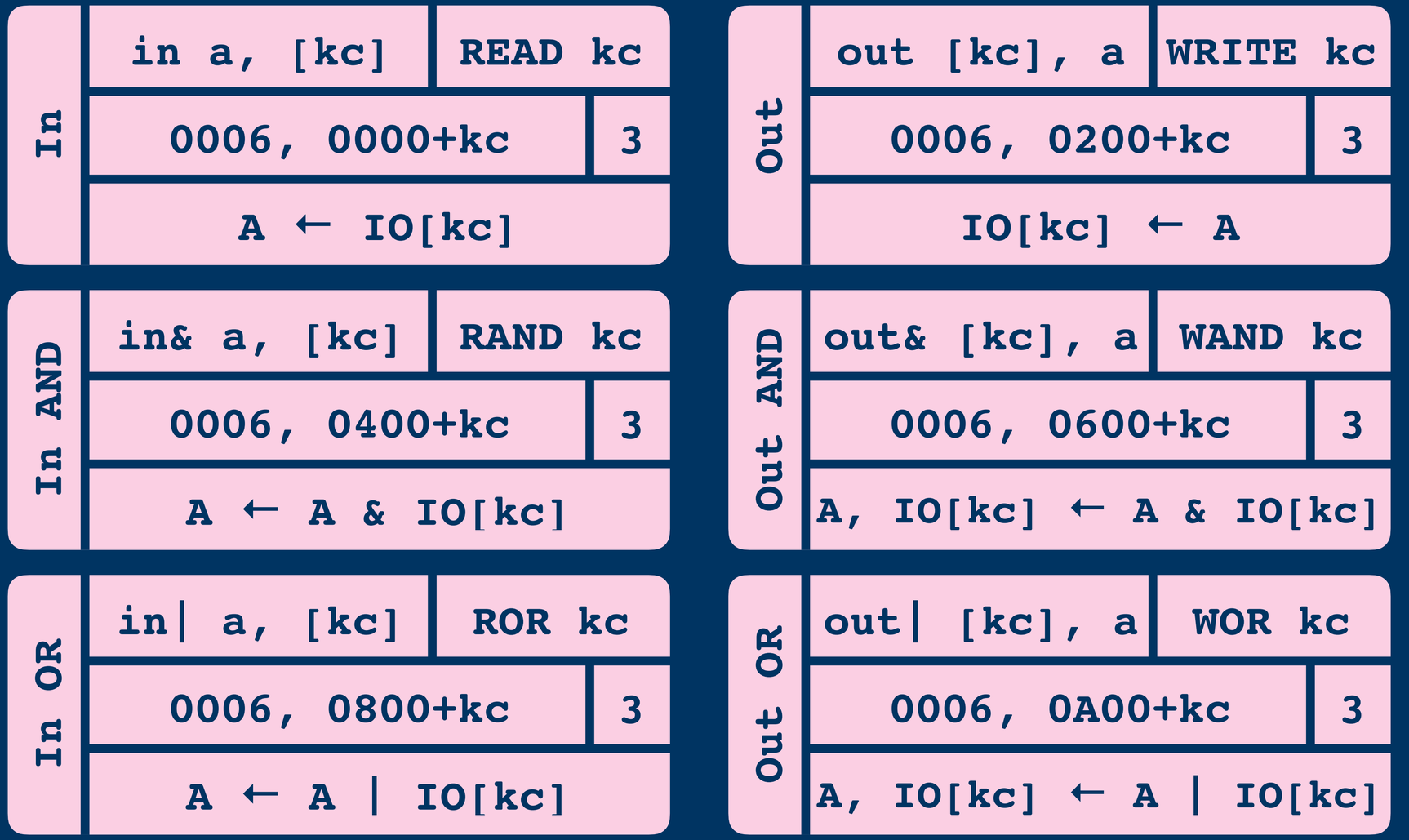
In AND		
in& a, [kc]	RAND kc	
0006, 0400+kc	3	
A ← A & IO[kc]		

Out AND		
out& [kc], a	WAND kc	
0006, 0600+kc	3	
A, IO[kc] ← A & IO[kc]		

In OR		
in| a, [kc]	ROR kc	
0006, 0800+kc	3	
A ← A | IO[kc]		

Out OR		
out| [kc], a	WOR kc	
0006, 0A00+kc	3	
A, IO[kc] ← A | IO[kc]		

I/O Channels


In
in a, [kc]
READ kc
0006, 0000+kc
3
A ← IO[kc]
Out
out [kc], a
WRITE kc
0006, 0200+kc
3
IO[kc] ← A
In AND
in& a, [kc]
RAND kc
0006, 0400+kc
3
A ← A & IO[kc]
Out AND
out& [kc], a
WAND kc
0006, 0600+kc
3
A, IO[kc] ← A & IO[kc]
In OR
in| a, [kc]
ROR kc
0006, 0800+kc
3
A ← A | IO[kc]
Out OR
out| [kc], a
WOR kc
0006, 0A00+kc
3
A, IO[kc] ← A | IO[kc]

I/O Channels

In		
in a, [kc]	READ kc	
0006, 0000+kc	3	
A ← IO[kc]		

Out		
out [kc], a	WRITE kc	
0006, 0200+kc	3	
IO[kc] ← A		

In AND		
in& a, [kc]	RAND kc	
0006, 0400+kc	3	
A ← A & IO[kc]		

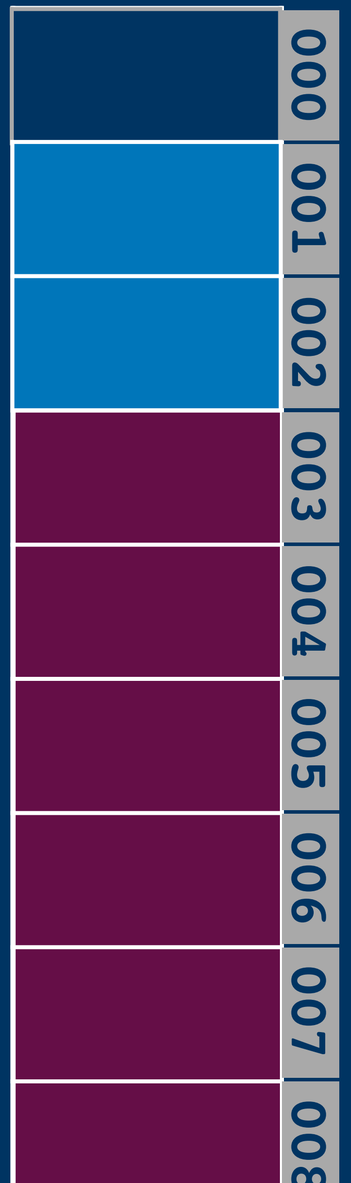
Out AND		
out& [kc], a	WAND kc	
0006, 0600+kc	3	
A, IO[kc] ← A & IO[kc]		

In OR		
in| a, [kc]	ROR kc	
0006, 0800+kc	3	
A ← A | IO[kc]		

Out OR		
out| [kc], a	WOR kc	
0006, 0A00+kc	3	
A, IO[kc] ← A | IO[kc]		

In XOR		
in^ a, [kc]	RXOR kc	
0006, 0C00+kc	3	
A ← A ^ IO[kc]		

I/O Channels



In		
in a, [kc]	READ kc	
0006, 0000+kc	3	
A ← IO[kc]		

Out		
out [kc], a	WRITE kc	
0006, 0200+kc	3	
IO[kc] ← A		

In AND		
in& a, [kc]	RAND kc	
0006, 0400+kc	3	
A ← A & IO[kc]		

Out AND		
out& [kc], a	WAND kc	
0006, 0600+kc	3	
A, IO[kc] ← A & IO[kc]		

In OR		
in| a, [kc]	ROR kc	
0006, 0800+kc	3	
A ← A | IO[kc]		

Out OR		
out| [kc], a	WOR kc	
0006, 0A00+kc	3	
A, IO[kc] ← A | IO[kc]		

In XOR		
in^ a, [kc]	RXOR kc	
0006, 0C00+kc	3	
A ← A ^ IO[kc]		

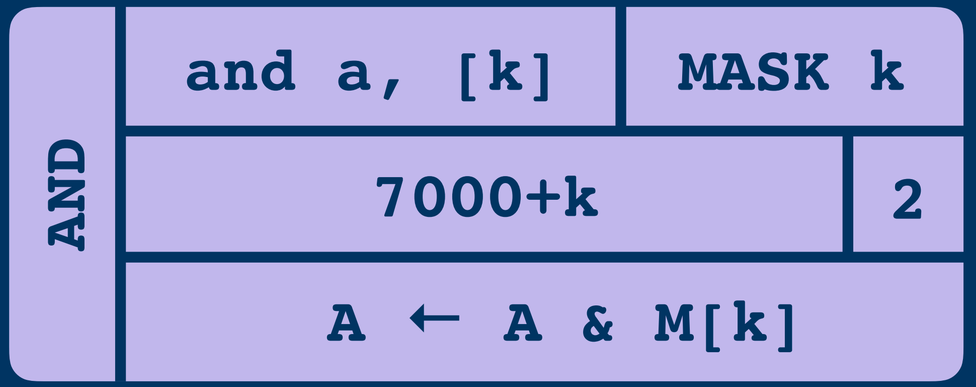
Boolean Logic

Boolean Logic

```
and a, b      ≡    in& a, [1]

or a, b       ≡    in| a, [1]

xor a, b      ≡    in^ a, [1]
```

Boolean Logic

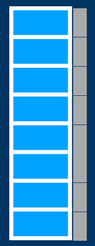
```
and a, b      ≡    in& a, [1]

or a, b       ≡    in| a, [1]

xor a, b      ≡    in^ a, [1]
```



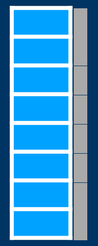
Counters


A
0007
000
B
0F00
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007

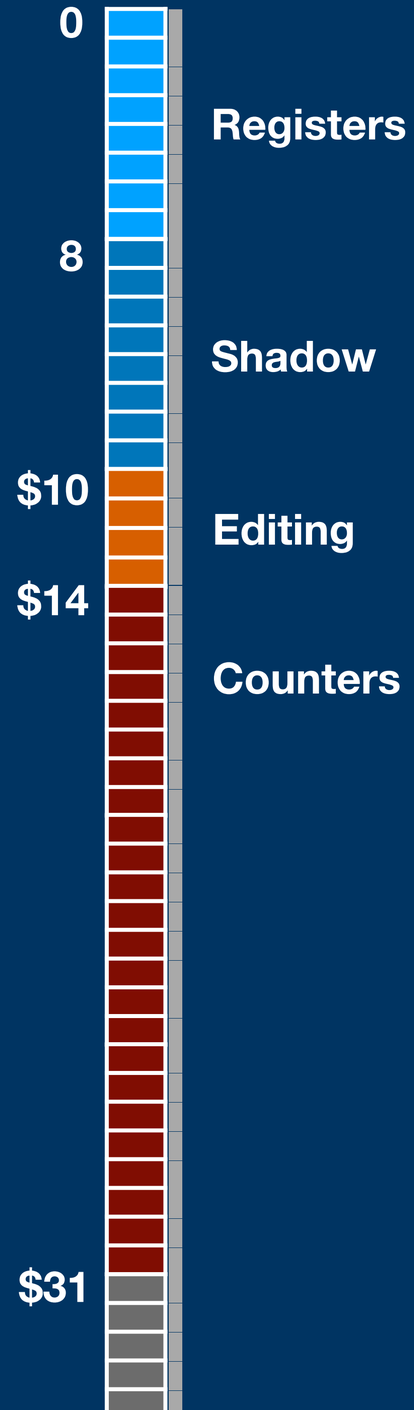
Counters



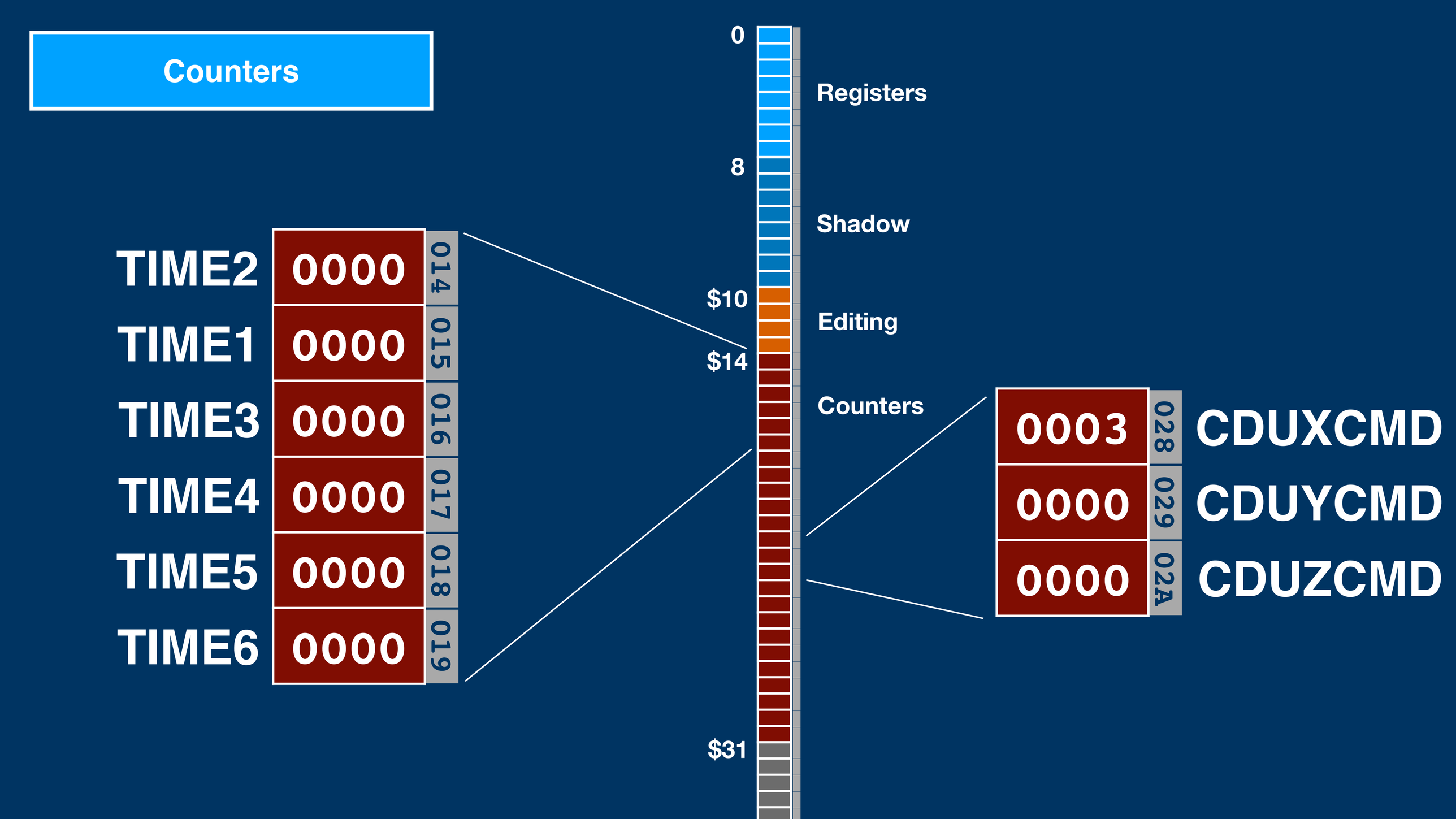
Counters



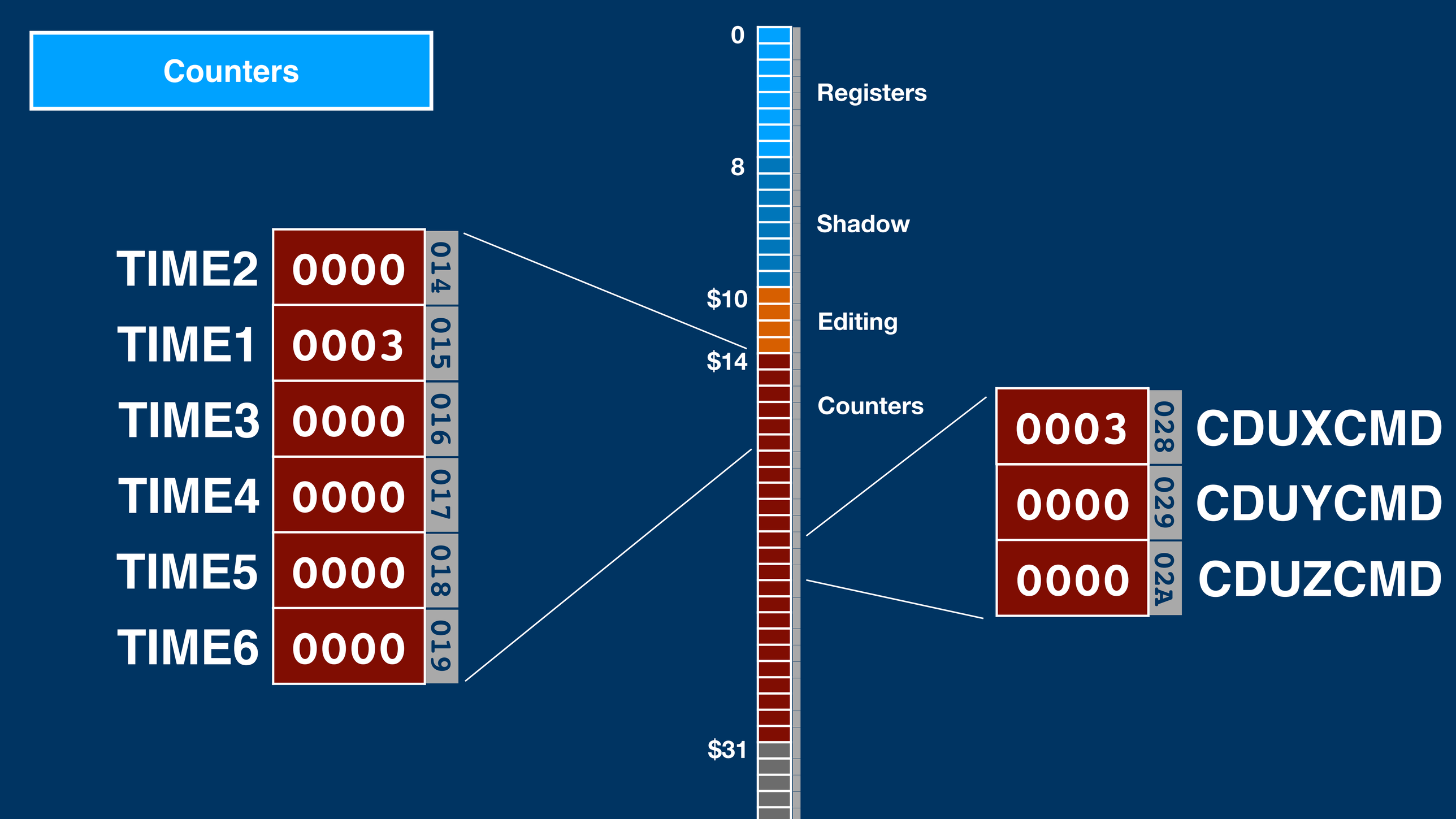
Counters



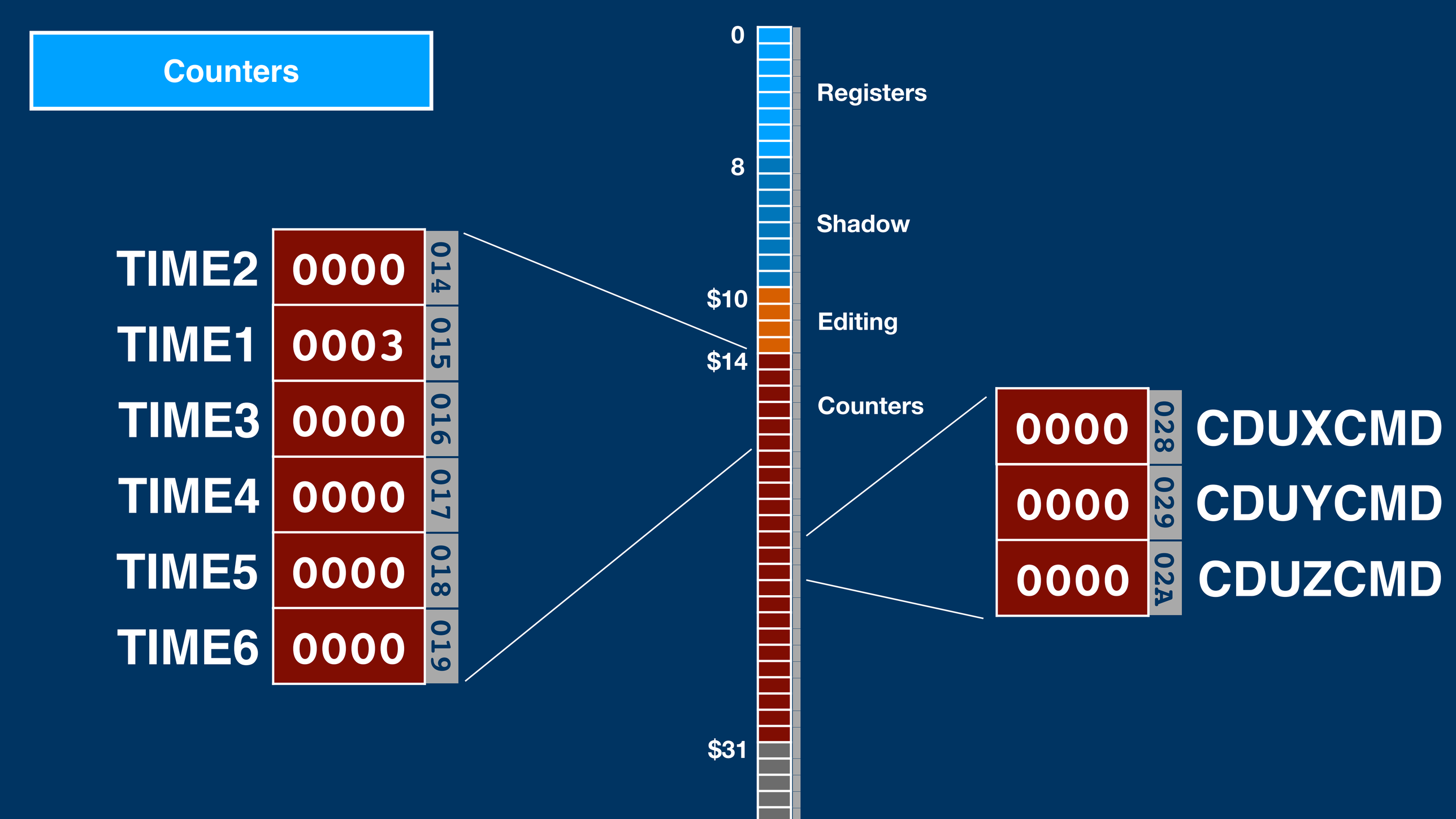
Counters


0
Registers
8
Shadow
$10
Editing
$14
Counters
$31
TIME2
0000
014
TIME1
0000
015
TIME3
0000
016
TIME4
0000
017
TIME5
0000
018
TIME6
0000
019
0003
028
CDUXCMD
0000
029
CDUYCMD
0000
02A
CDUZCMD

Counters


0
Registers
8
Shadow
$10
Editing
$14
Counters
$31
TIME2
0000
014
TIME1
0003
015
TIME3
0000
016
TIME4
0000
017
TIME5
0000
018
TIME6
0000
019
0003
028
CDUXCMD
0000
029
CDUYCMD
0000
02A
CDUZCMD

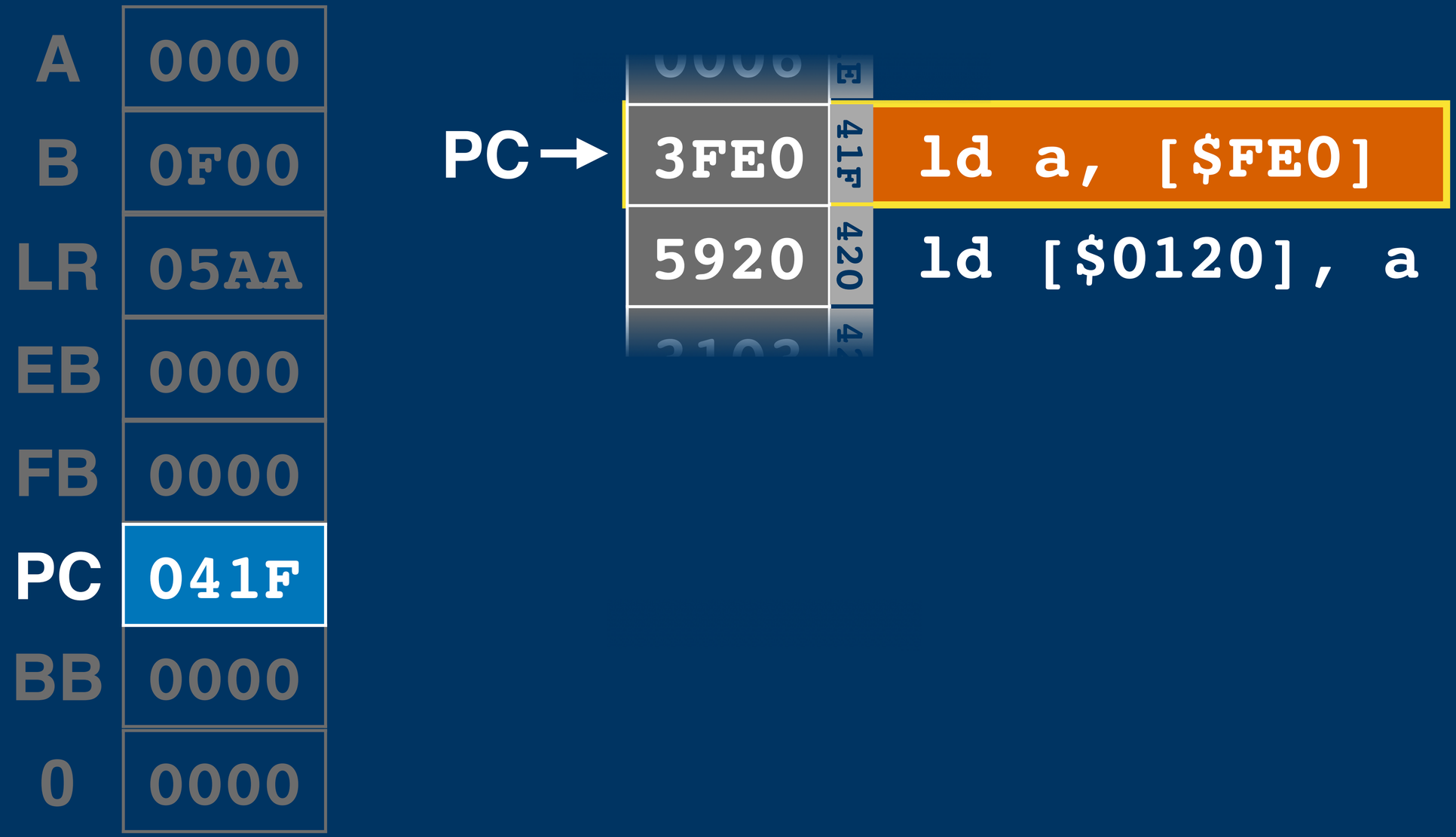
Counters


0
Registers
8
Shadow
$10
Editing
$14
Counters
$31
TIME2
0000
014
TIME1
0003
015
TIME3
0000
016
TIME4
0000
017
TIME5
0000
018
TIME6
0000
019
0000
028
CDUXCMD
0000
029
CDUYCMD
0000
02A
CDUZCMD

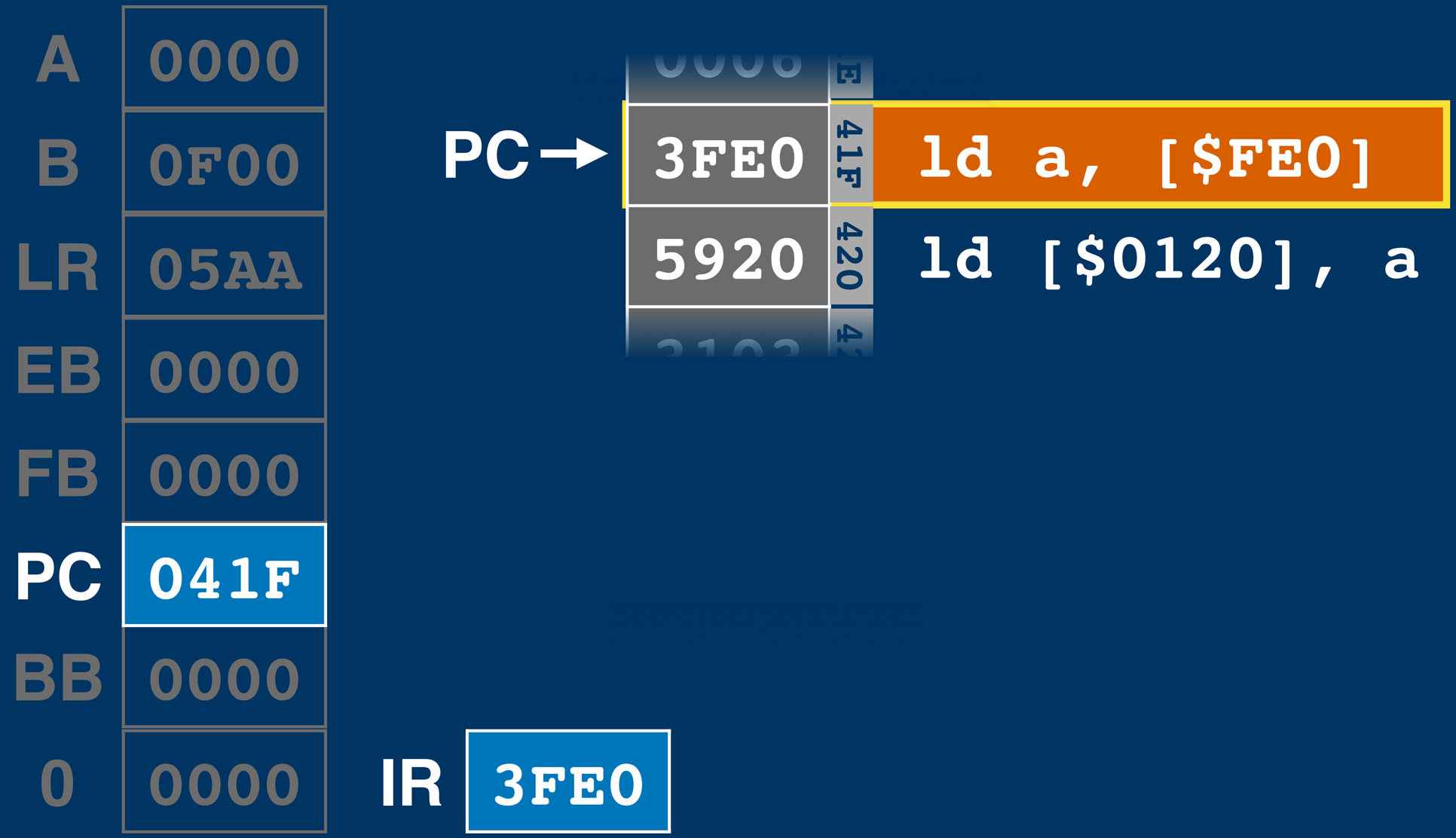
Interrupts

A	0000
B	0F00
LR	05AA
EB	0000
FB	0000
PC	041F
BB	0000
0	0000

Interrupts


A
0000
B
0F00
LR
05AA
EB
0000
FB
0000
PC
041F
BB
0000
0
0000
PC
3FE0
41F
ld a, [$FE0]
5920
420
ld [$0120], a

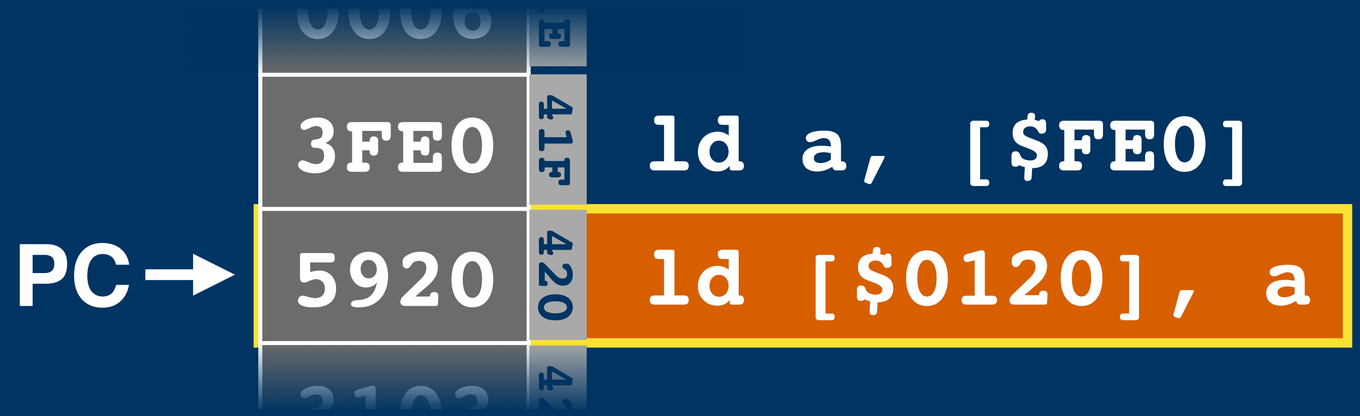
Interrupts


A
0000
B
0F00
LR
05AA
EB
0000
FB
0000
PC
041F
BB
0000
0
0000
IR
3FE0
PC
3FE0
41F
ld a, [$FE0]
5920
420
ld [$0120], a

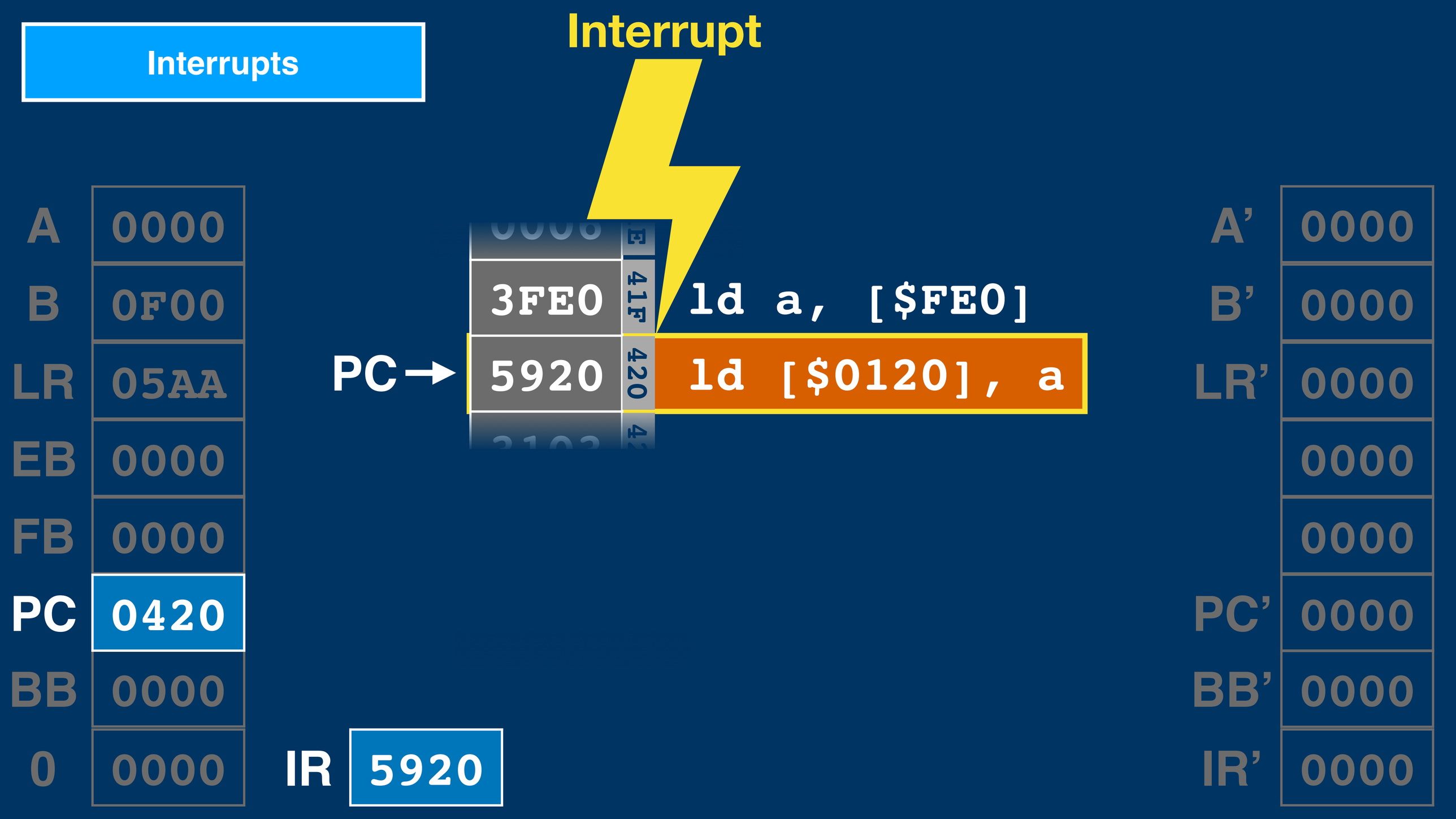
Interrupts

A	0000
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0420
BB	0000
0	0000

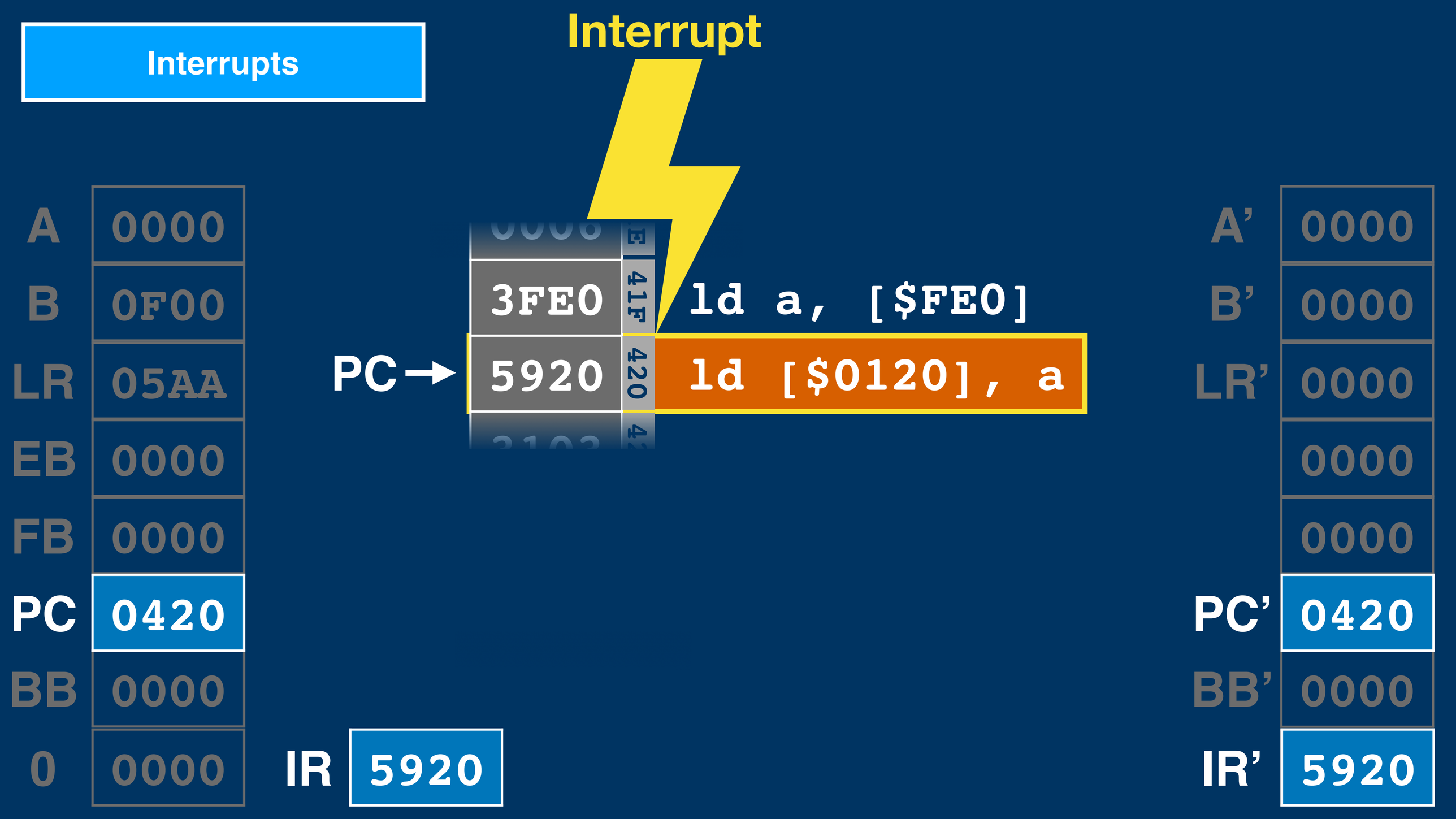
IR 5920



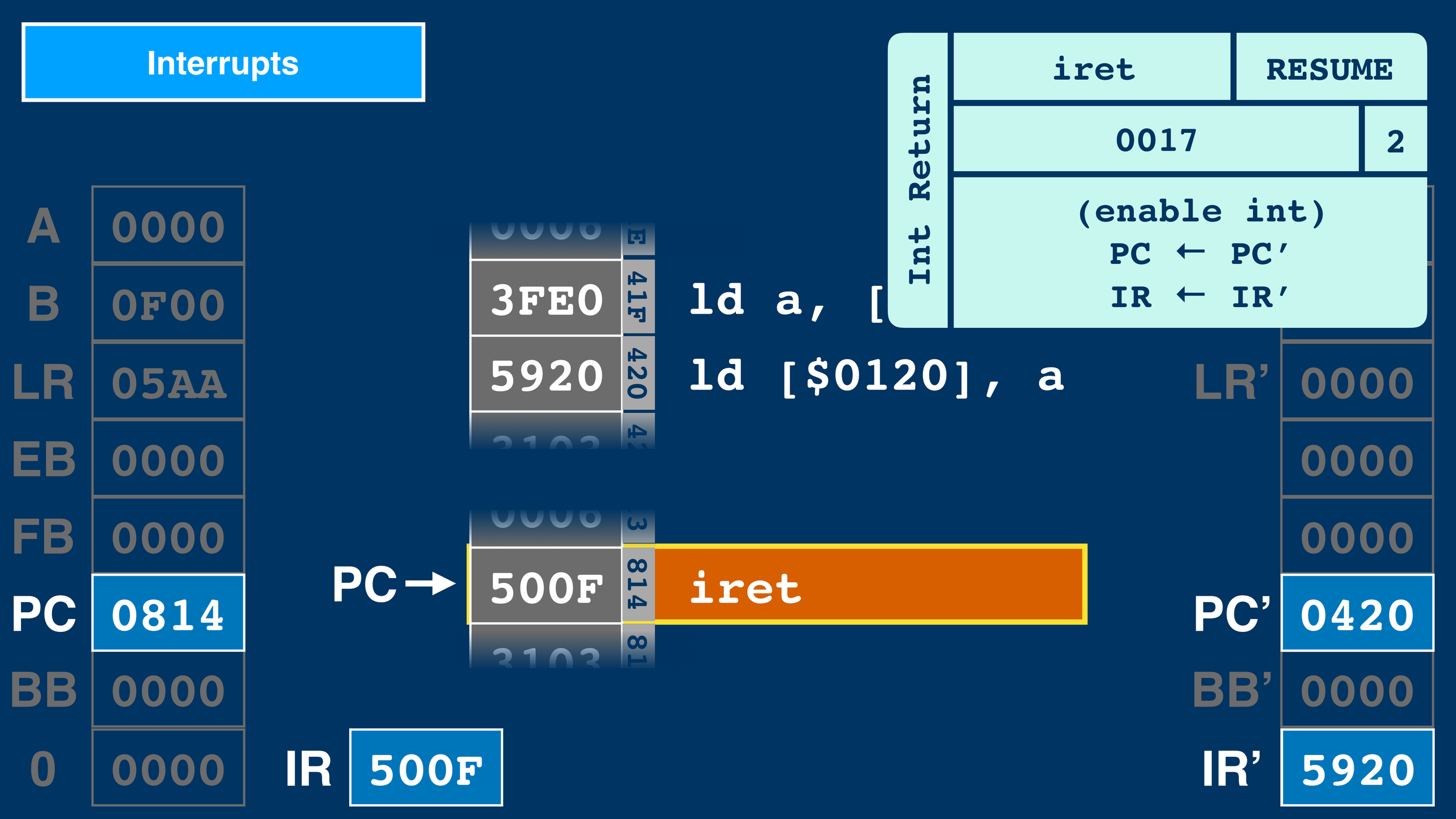
Interrupts


Interrupt
A 0000
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0420
BB 0000
0 0000
IR 5920
PC→
3FE0 41F
5920 420
ld a, [$FE0]
ld [$0120], a
A' 0000
B' 0000
LR' 0000
0000
0000
PC' 0000
BB' 0000
IR' 0000

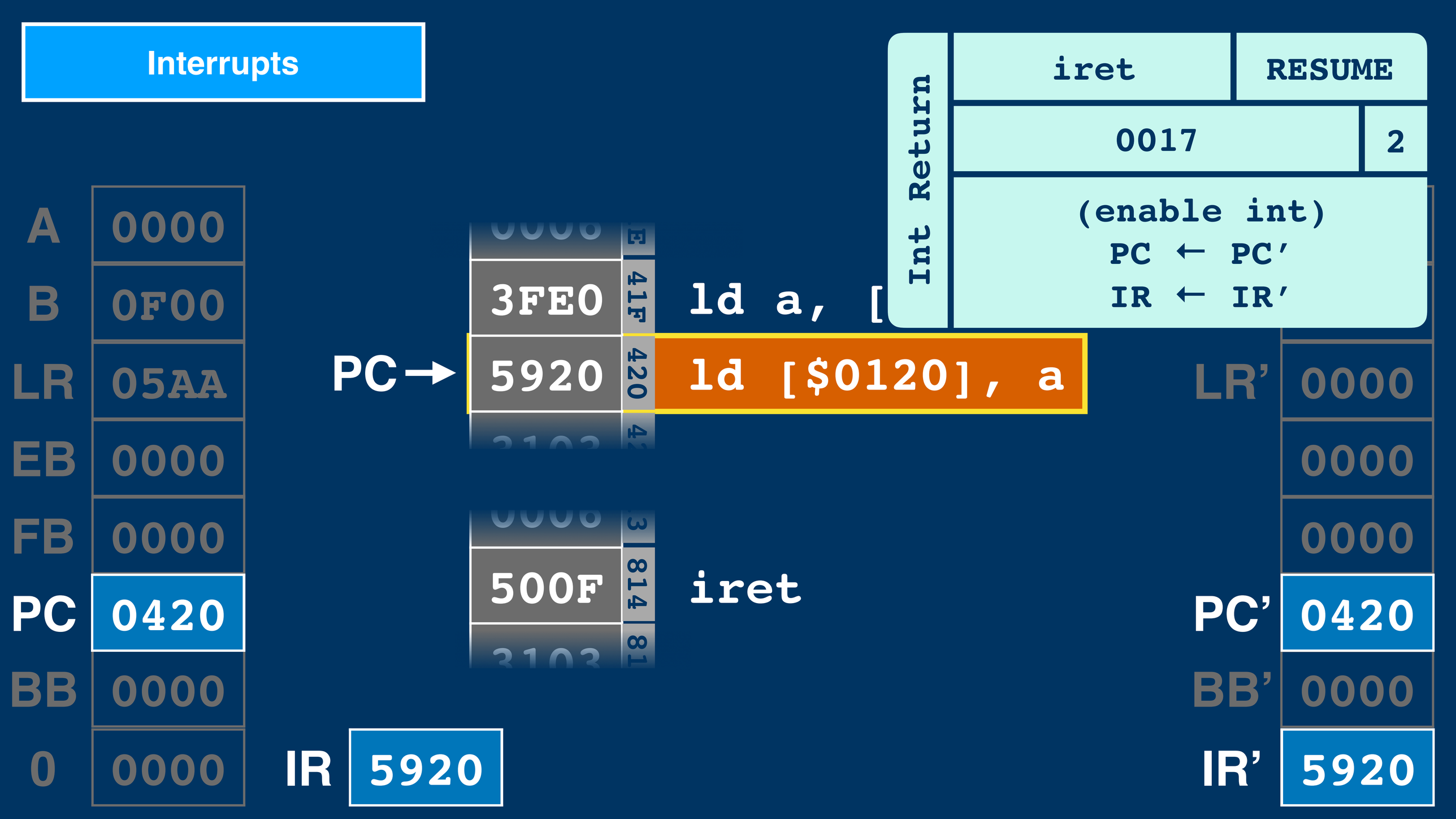
Interrupts


Interrupt
A 0000
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0420
BB 0000
0 0000
IR 5920
3FE0 41F
PC → 5920 420
ld a, [$FE0]
ld [$0120], a
A’ 0000
B’ 0000
LR’ 0000
0000
0000
PC’ 0420
BB’ 0000
IR’ 5920

Interrupts


Int Return
iret
RESUME
0017
2
(enable int)
PC ← PC'
IR ← IR'
A 0000
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0814
BB 0000
0 0000
IR 500F
3FE0 41F ld a, [
5920 420 ld [$0120], a
PC→
500F 814 iret
LR' 0000
0000
0000
PC' 0420
BB' 0000
IR' 5920

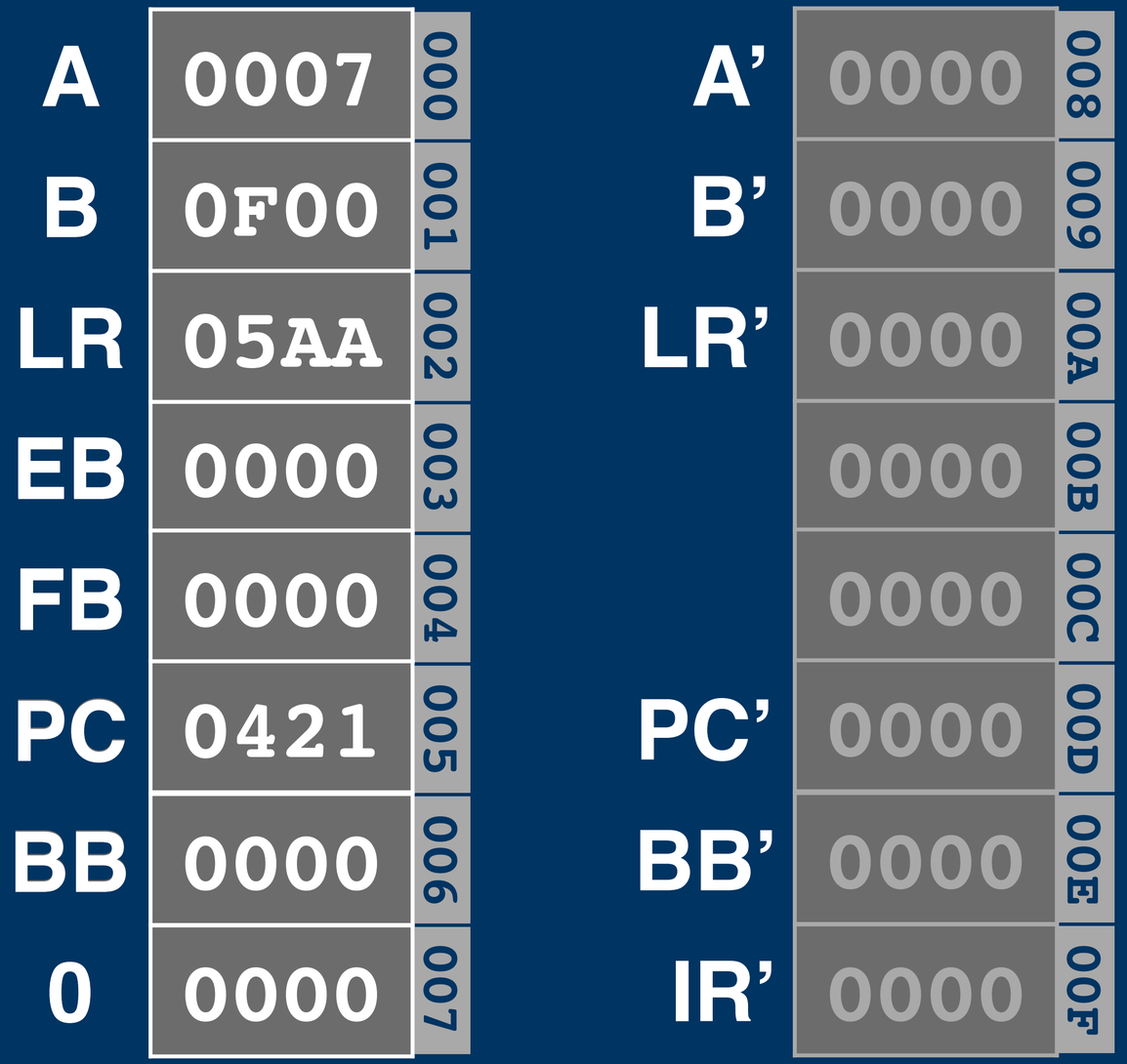
Interrupts


Int Return
iret
RESUME
0017
2
(enable int)
PC ← PC'
IR ← IR'
A 0000
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0420
BB 0000
0 0000
IR 5920
PC
3FE0 41F ld a, [
5920 420 ld [$0120], a
500F 814 iret
LR' 0000
0000
0000
PC' 0420
BB' 0000
IR' 5920

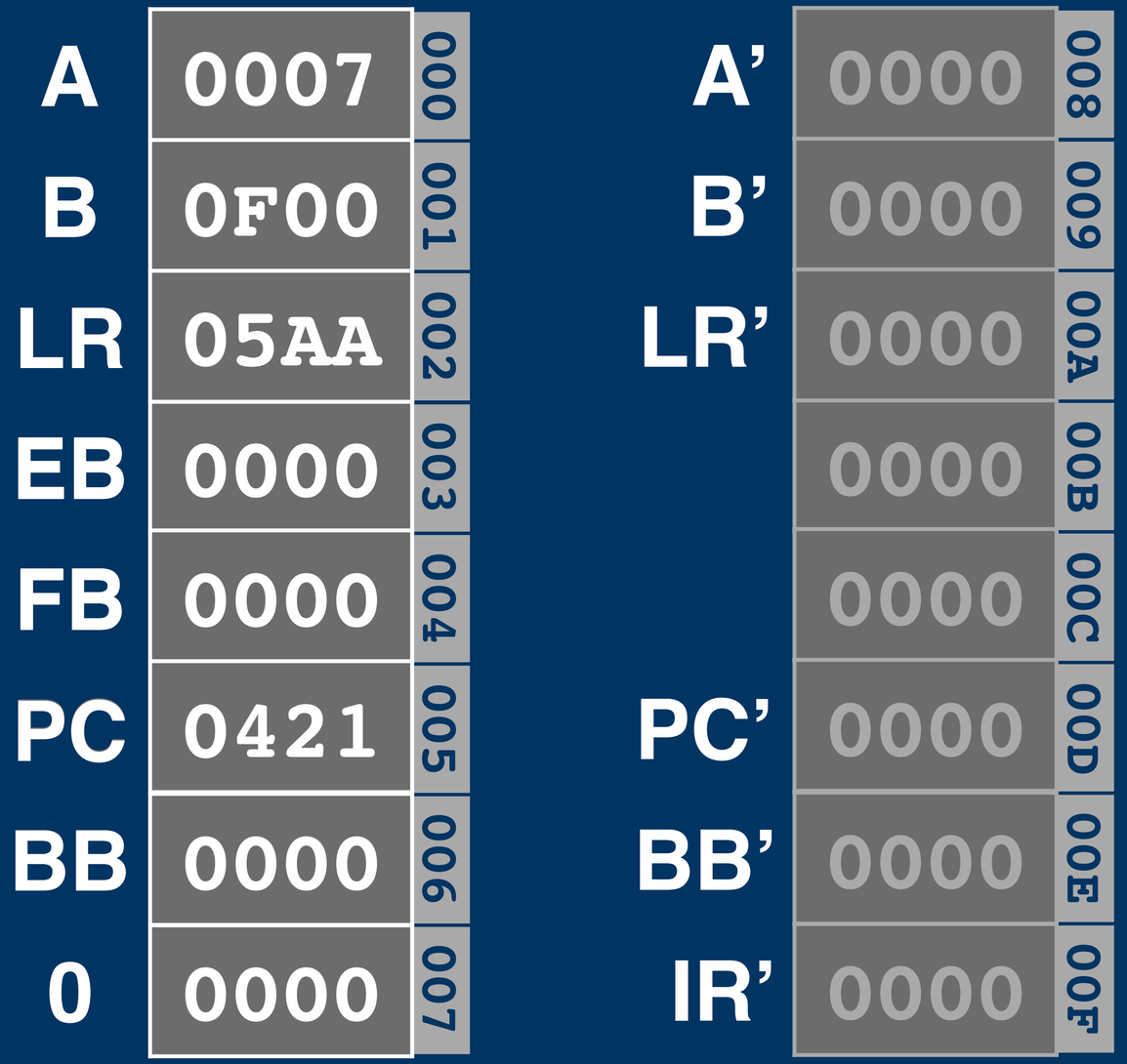
Interrupts


A
0007
000
B
0F00
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007

Interrupts

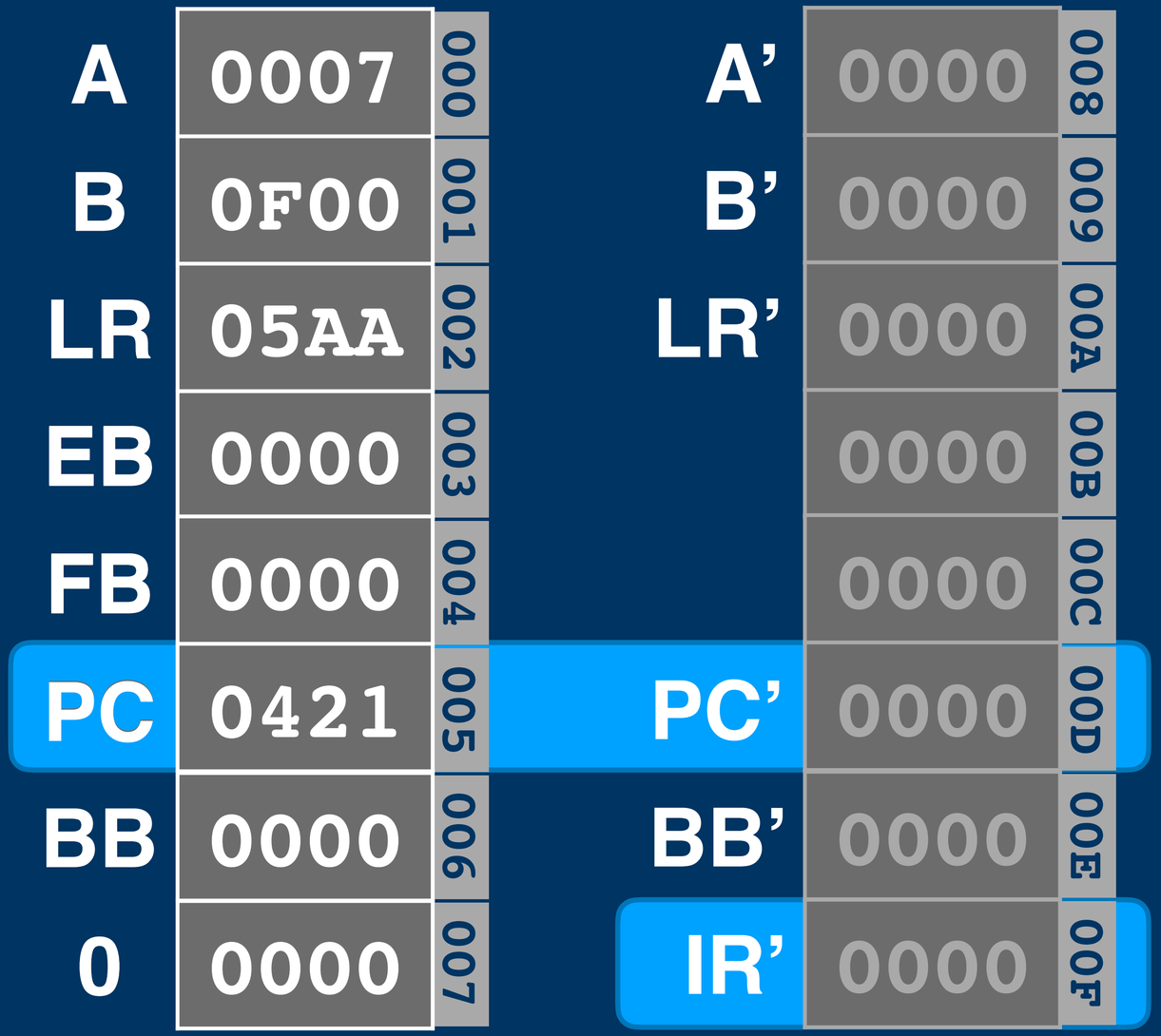

A
0007
000
B
0F00
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007
A’
0000
008
B’
0000
009
LR’
0000
00A
0000
00B
0000
00C
PC’
0000
00D
BB’
0000
00E
IR’
0000
00F

Interrupts


A
0007
000
B
0F00
001
LR
05AA
002
EB
0000
003
FB
0000
004
PC
0421
005
BB
0000
006
0
0000
007
A’
0000
008
B’
0000
009
LR’
0000
00A
0000
00B
0000
00C
PC’
0000
00D
BB’
0000
00E
IR’
0000
00F

Interrupts

Saved
&
Restored
in *Hardware*

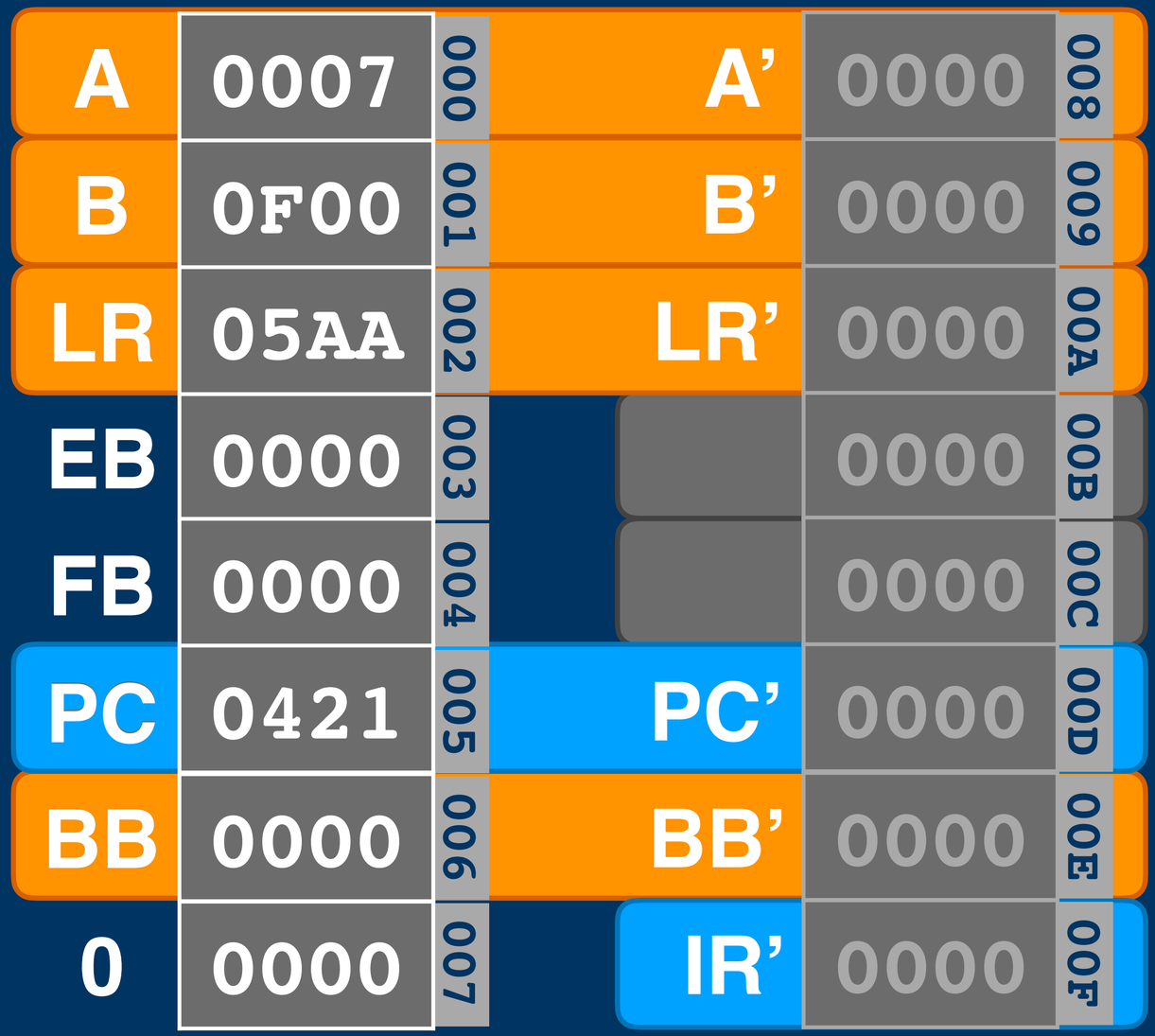


Interrupts

Saved & Restored in *Hardware*

Reserved for Saving & Restoring in *Software*

Unused

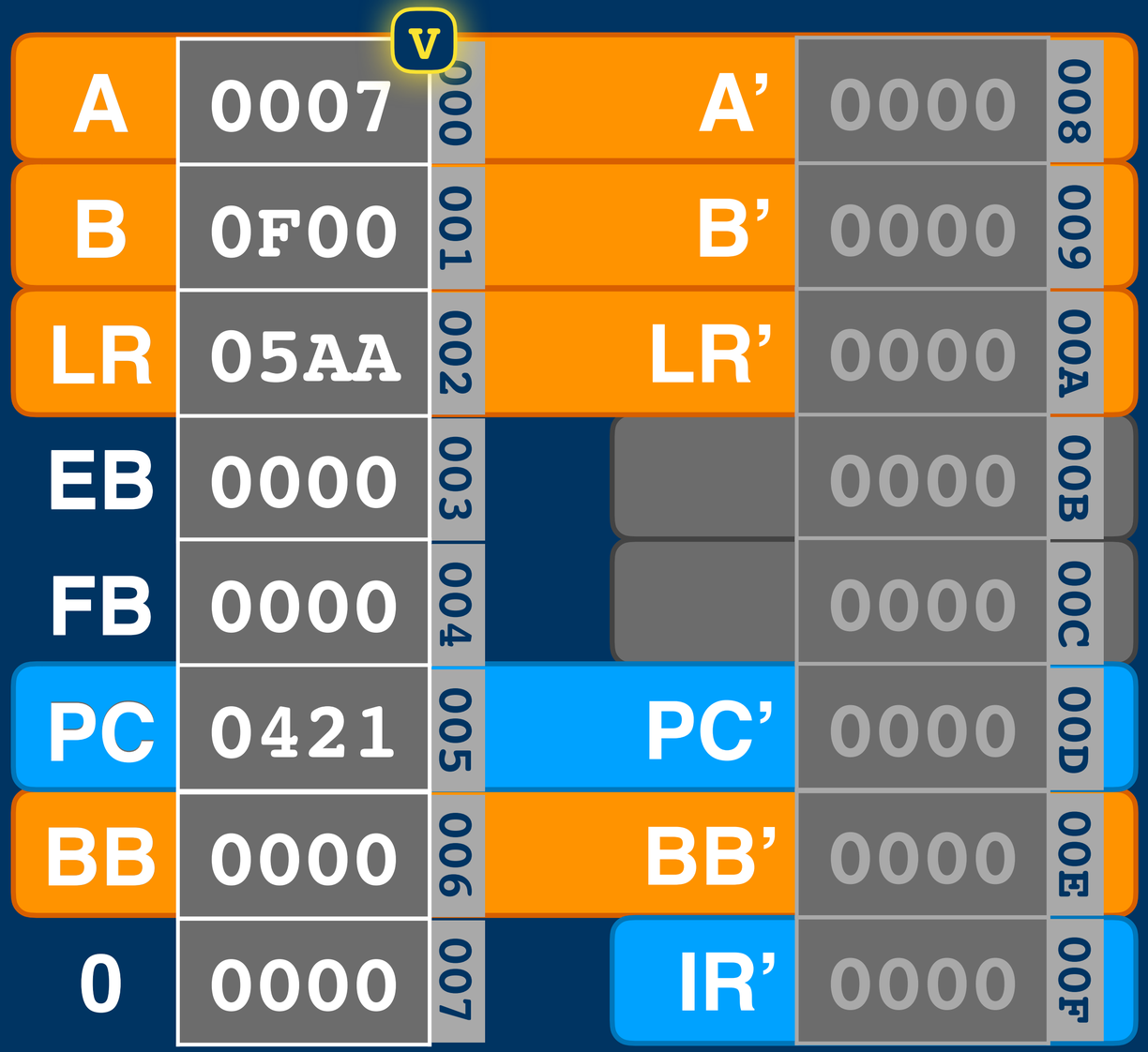


Interrupts

Saved & Restored in *Hardware*

Reserved for Saving & Restoring in *Software*

Unused



Interrupts & Overflow

A	0007	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

PC →

```
ld a, [$200]
add a, [$201]
nop
nop
ld [$202], a
jmp cont
ld [$203], a
```

Interrupts & Overflow

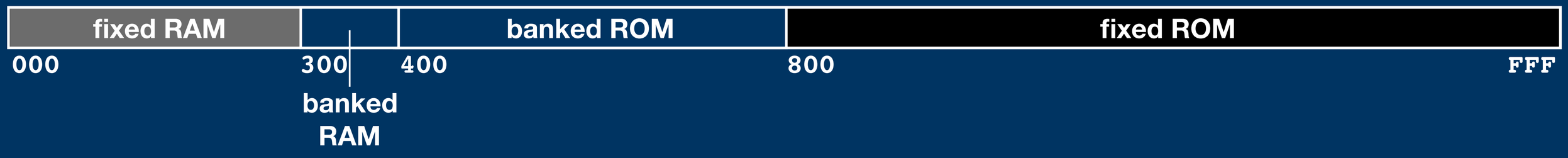
A	0000	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

```
ld a, [$200]
add a, [$201]
nop
nop
ld [$202], a
jmp cont
ld [$203], a
```

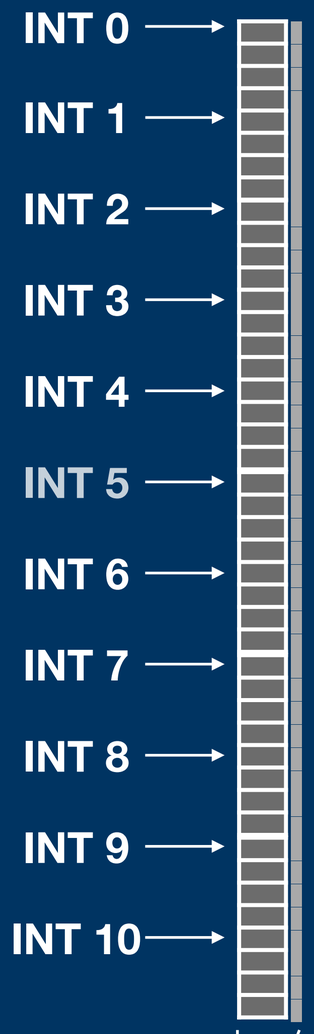
PC →

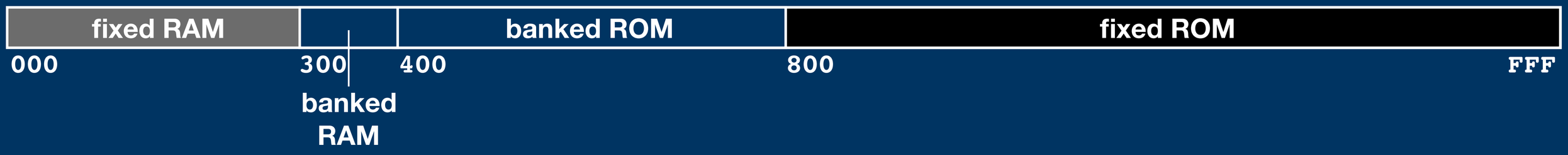
Interrupts disabled

Interrupts

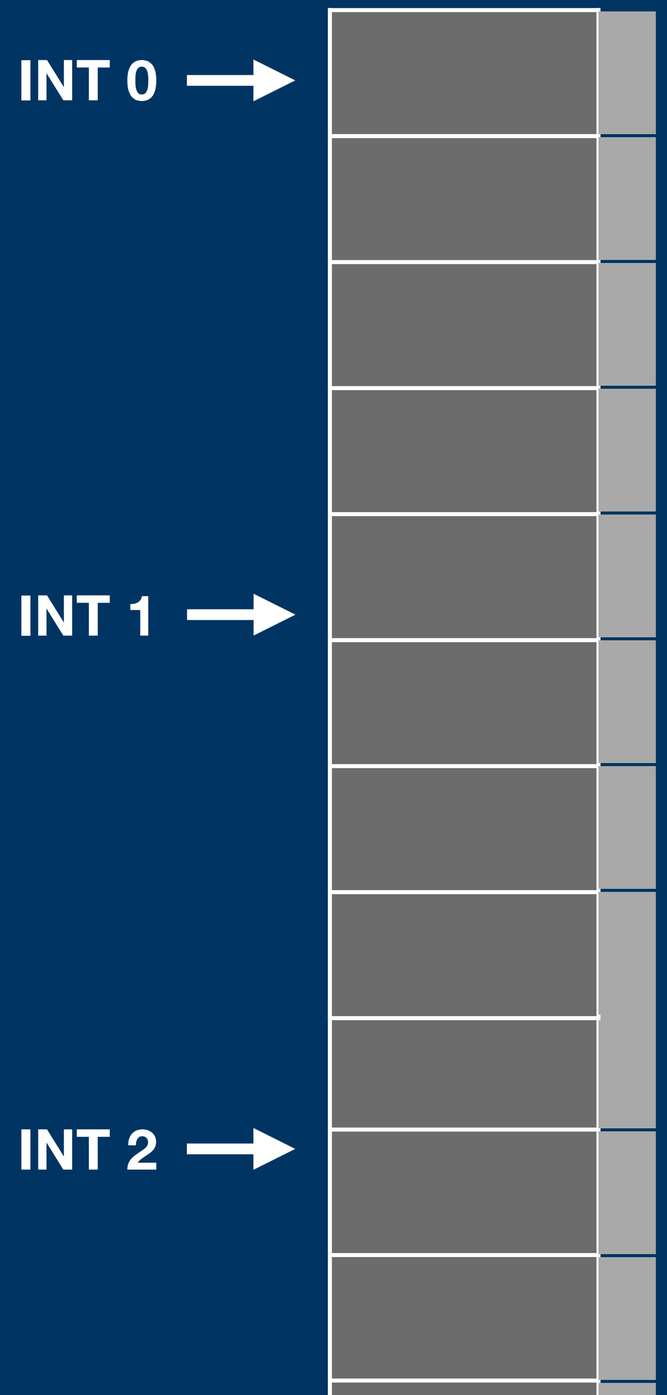


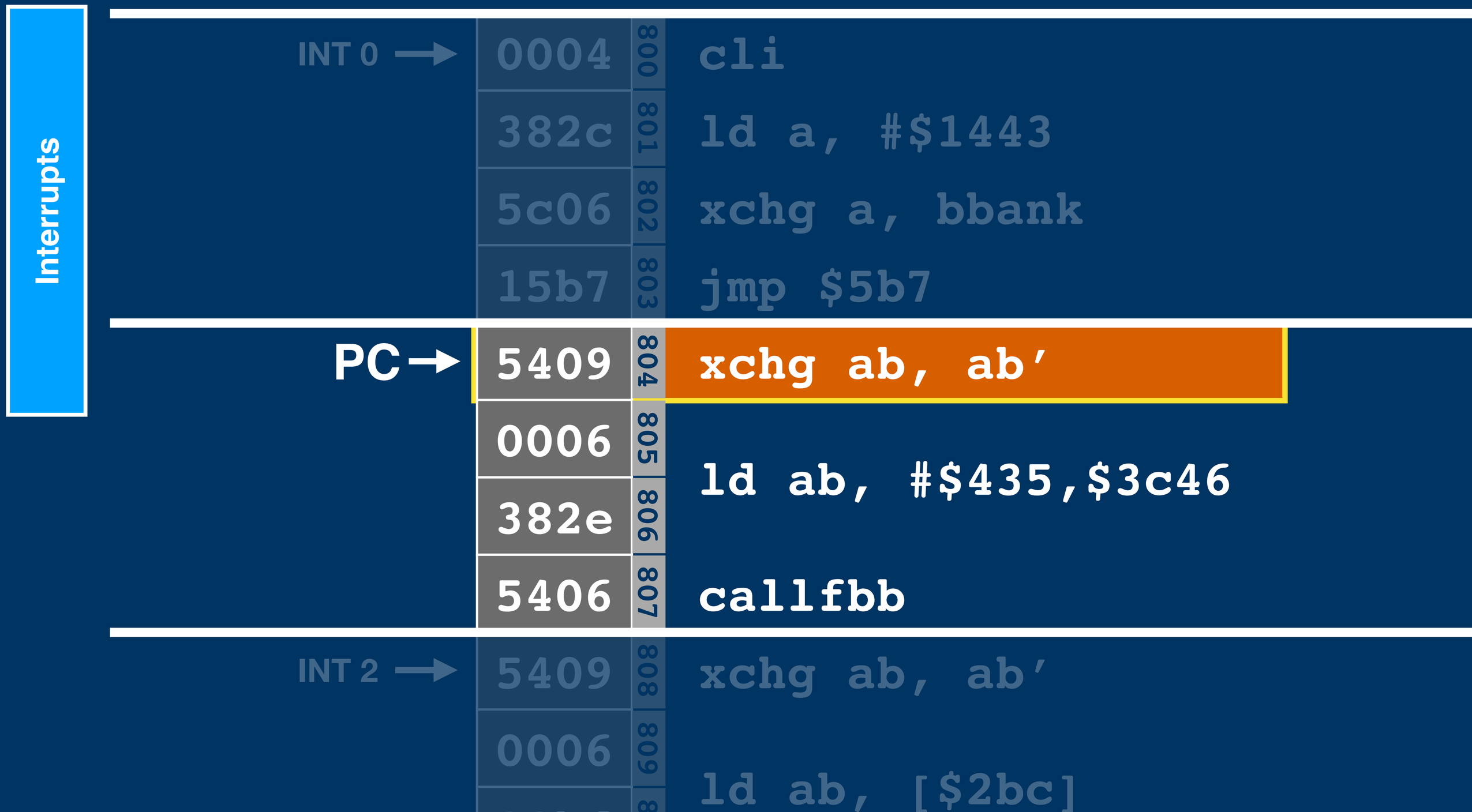
Interrupts


INT 0
INT 1
INT 2
INT 3
INT 4
INT 5
INT 6
INT 7
INT 8
INT 9
INT 10

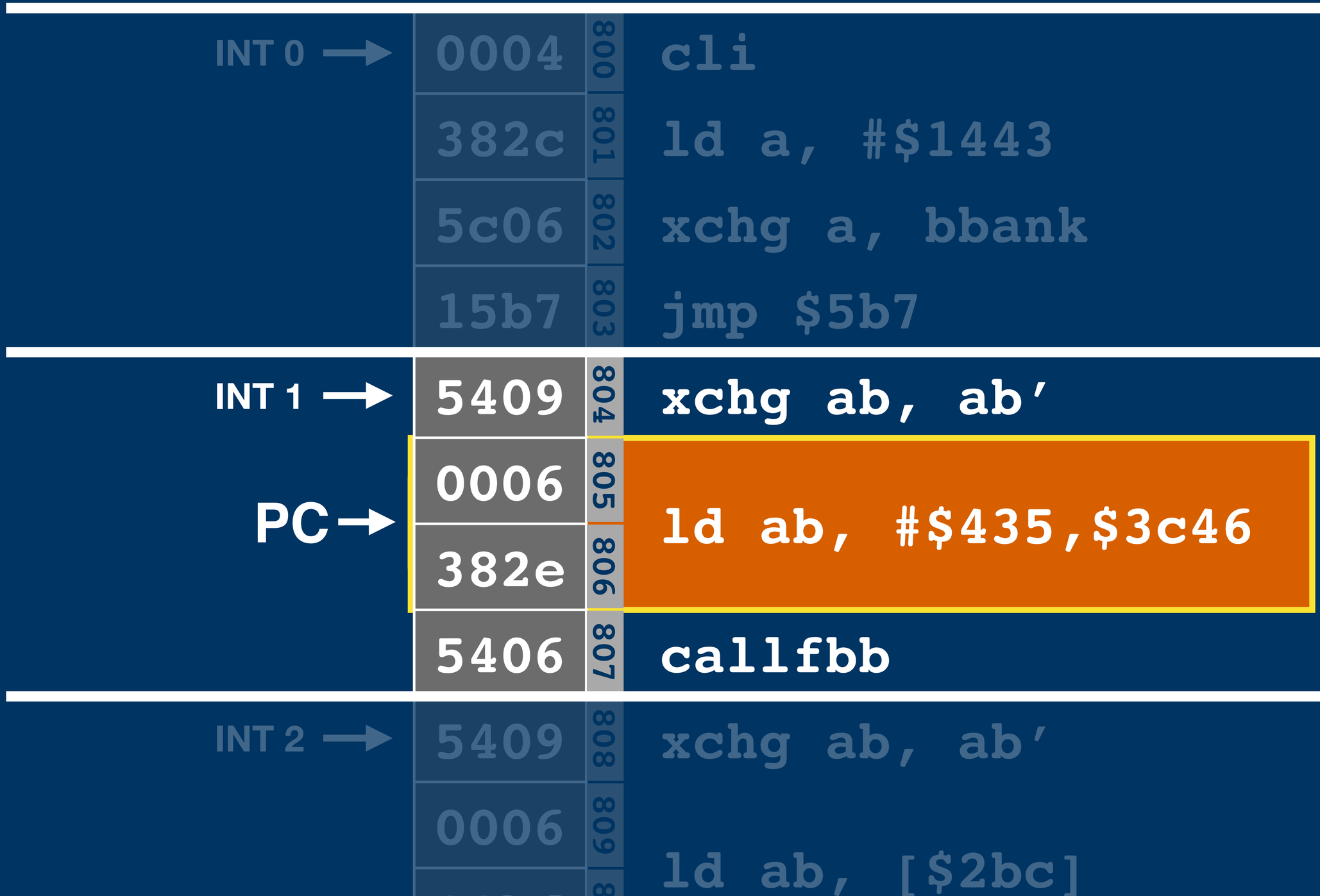

fixed RAM
banked ROM
fixed ROM
000
300
400
800
FFF
banked
RAM

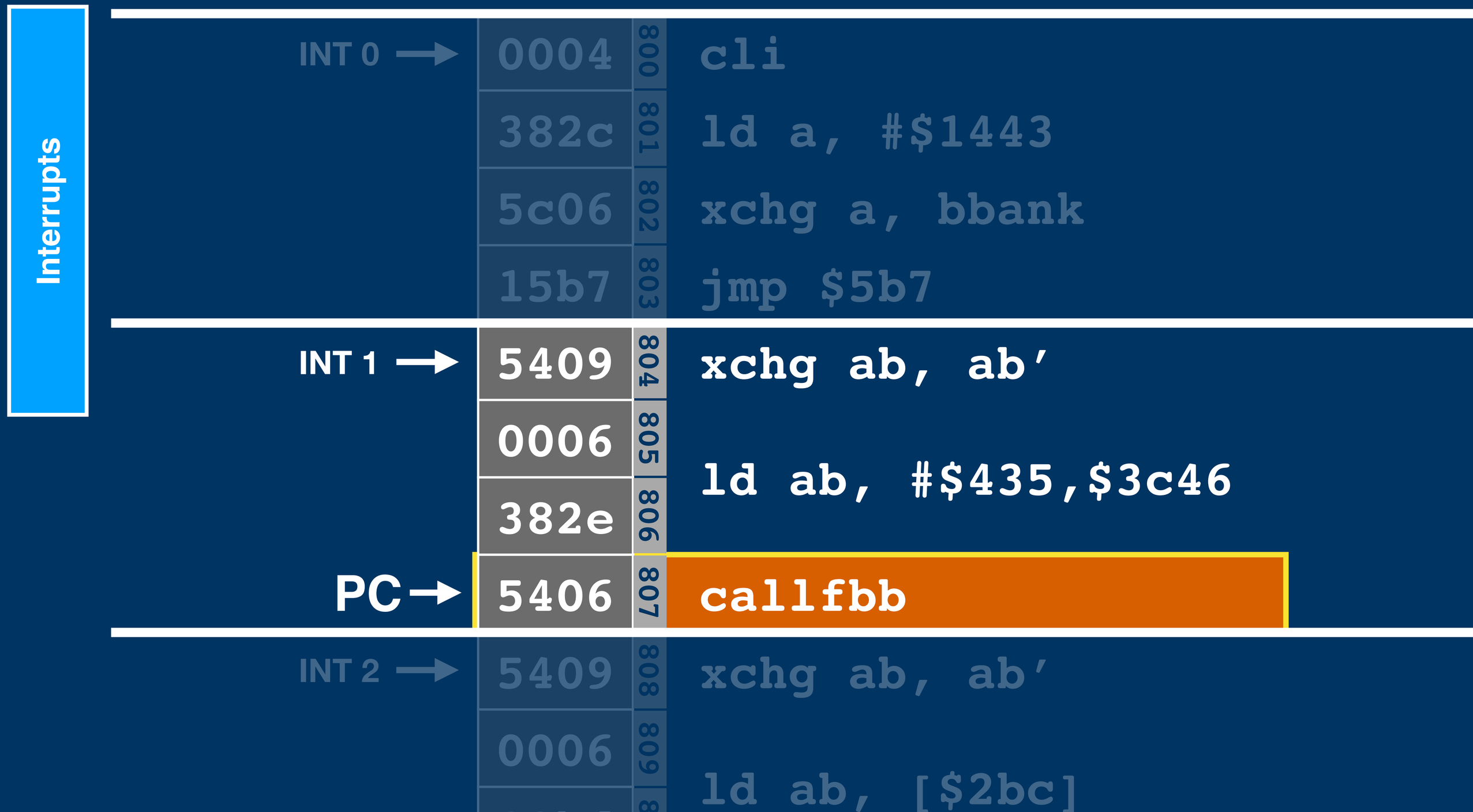
Interrupts




Interrupts
INT 0
0004
800
cli
382c
801
ld a, #$1443
5c06
802
xchg a, bbank
15b7
803
jmp $5b7
PC
5409
804
xchg ab, ab'
0006
805
ld ab, #$435,$3c46
382e
806
5406
807
callfbb
INT 2
5409
808
xchg ab, ab'
0006
809
ld ab, [$2bc]

Interrupts

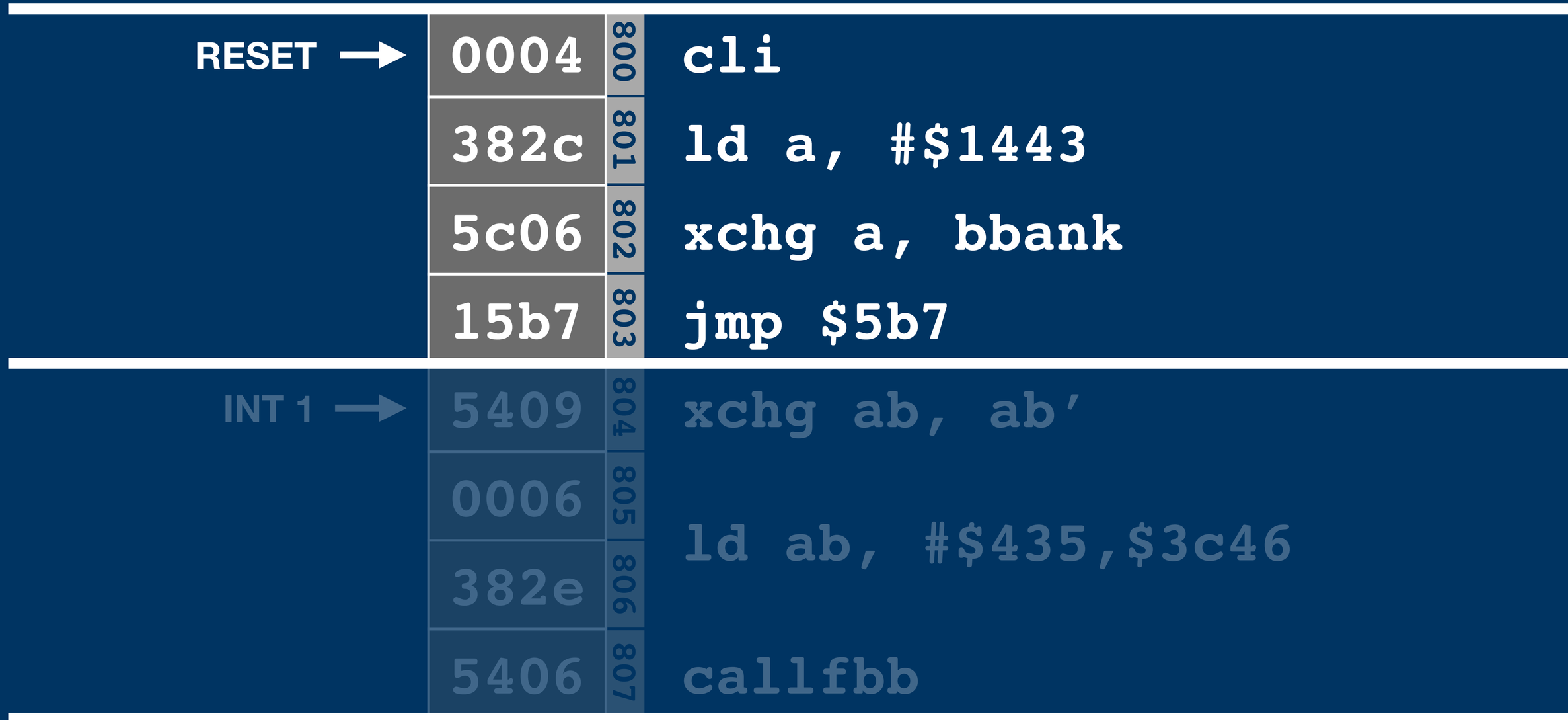

INT 0
0004
800
cli
382c
801
ld a, #$1443
5c06
802
xchg a, bbank
15b7
803
jmp $5b7
INT 1
5409
804
xchg ab, ab'
PC
0006
805
ld ab, #$435,$3c46
382e
806
5406
807
callfbb
INT 2
5409
808
xchg ab, ab'
0006
809
ld ab, [$2bc]


Interrupts
INT 0
0004
800
cli
382c
801
ld a, #$1443
5c06
802
xchg a, bbank
15b7
803
jmp $5b7
INT 1
5409
804
xchg ab, ab'
0006
805
ld ab, #$435,$3c46
382e
806
PC
5406
807
callfbb
INT 2
5409
808
xchg ab, ab'
0006
809
ld ab, [$2bc]

Interrupts

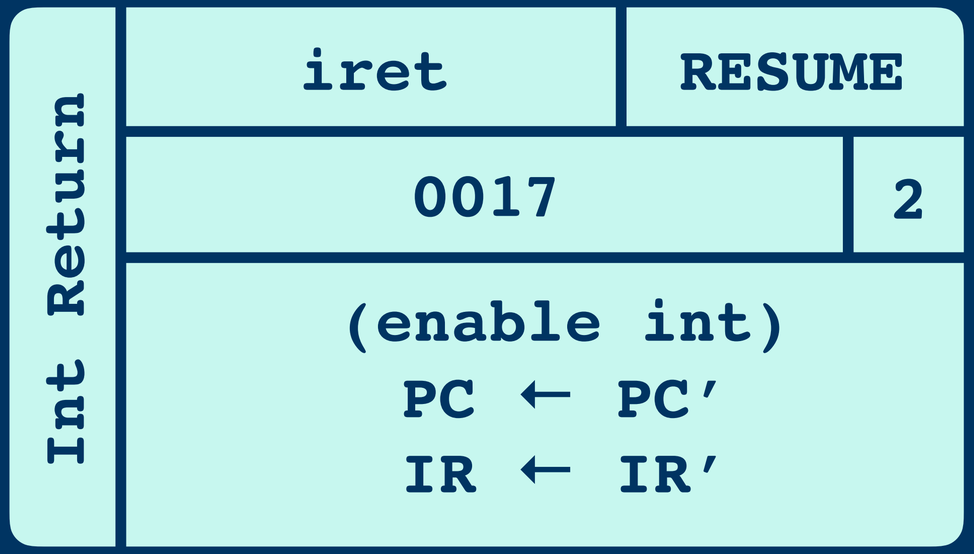
INT 0 →	0004	800	cli
	382c	801	ld a, #$1443
	5c06	802	xchg a, bbank
	15b7	803	jmp $5b7
INT 1 →	5409	804	xchg ab, ab’
	0006	805	ld ab, #$435,$3c46
	382e	806	
	5406	807	callfbb

Interrupts

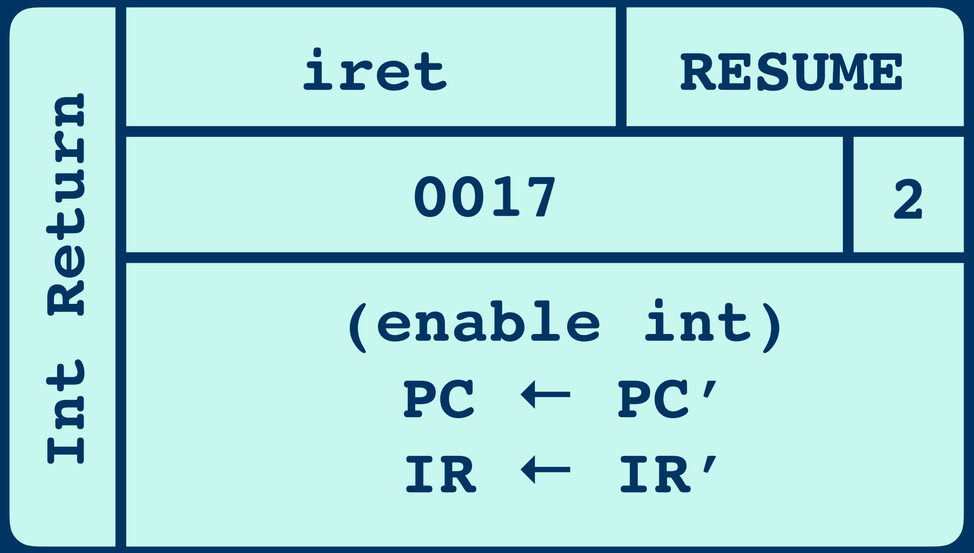

RESET
0004
800
cli
382c
801
ld a, #$1443
5c06
802
xchg a, bbank
15b7
803
jmp $5b7
INT 1
5409
804
xchg ab, ab'
0006
805
ld ab, #$435,$3c46
382e
806
5406
807
callfbb

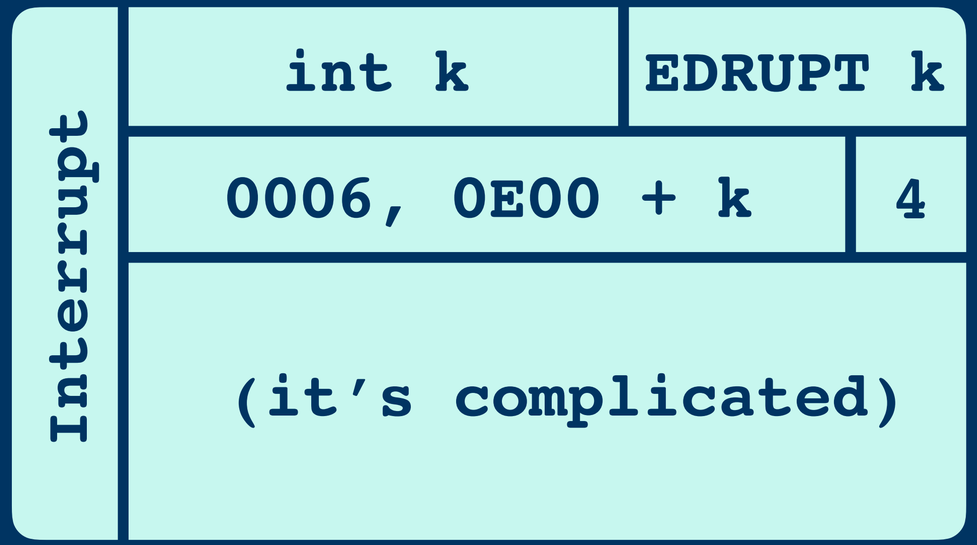
Interrupts

Interrupts

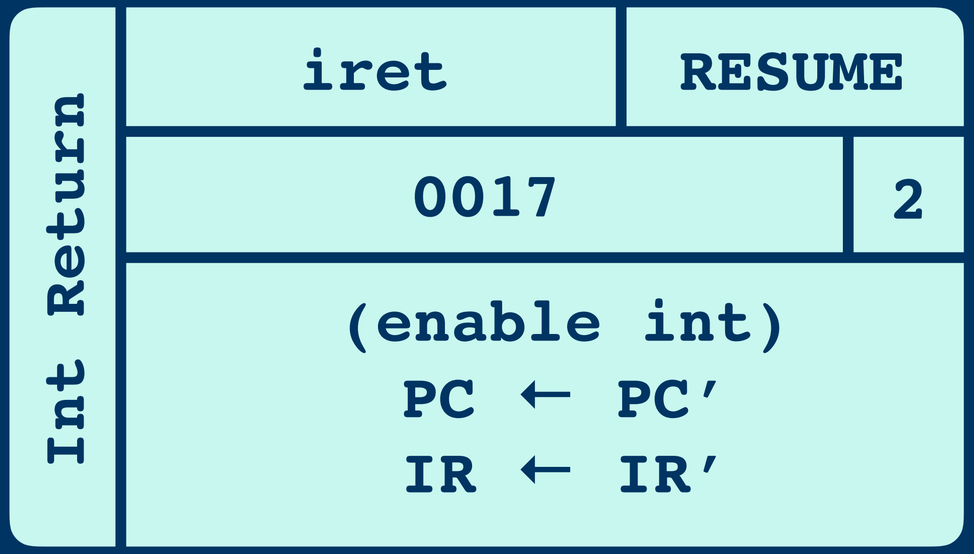


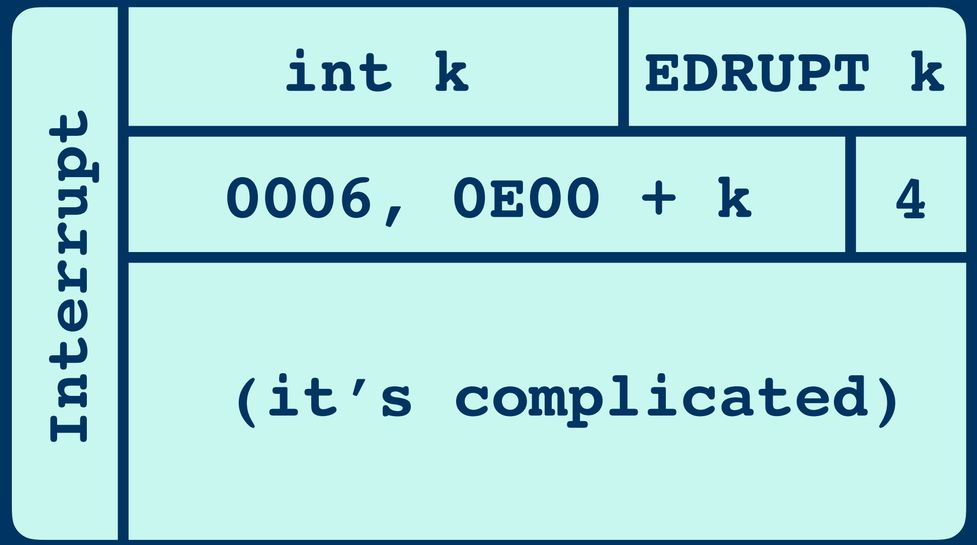
Interrupts

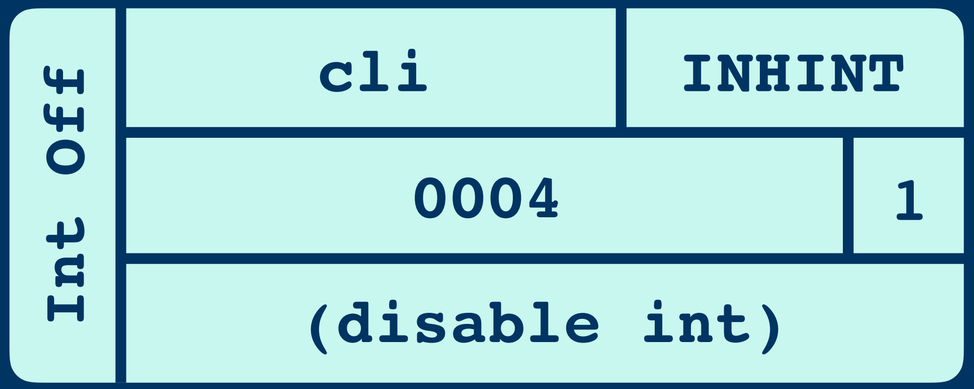

Int Return
iret
RESUME
0017
2
(enable int)
PC ← PC′
IR ← IR′

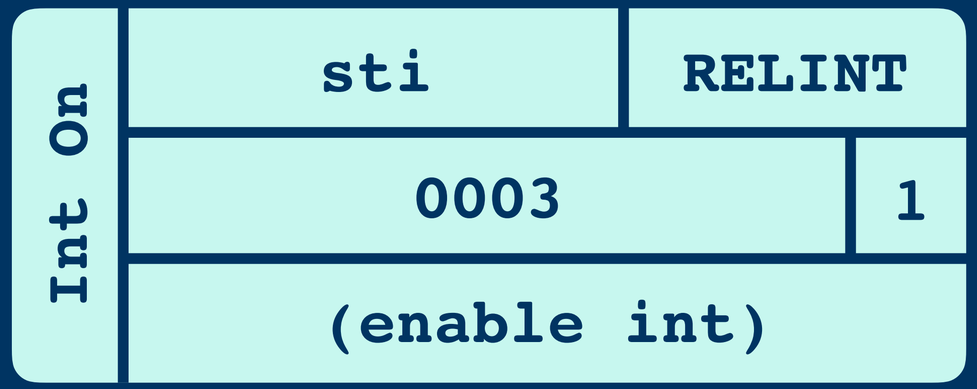

Interrupt
int k
EDRUPT k
0006, 0E00 + k
4
(it’s complicated)

Interrupts

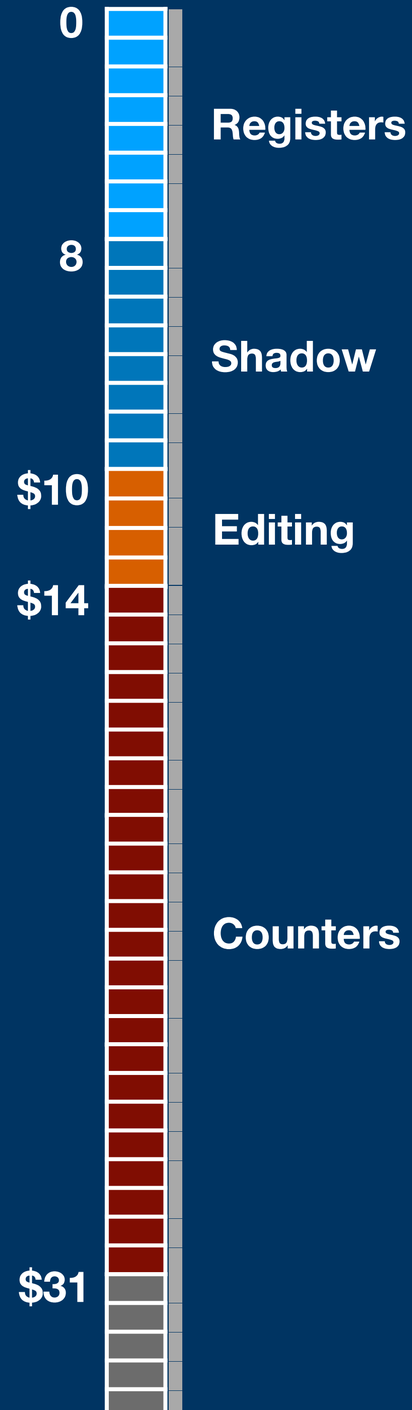

Int Return
iret
RESUME
0017
2
(enable int)
PC ← PC'
IR ← IR'


Interrupt
int k
EDRUPT k
0006, 0E00 + k
4
(it's complicated)

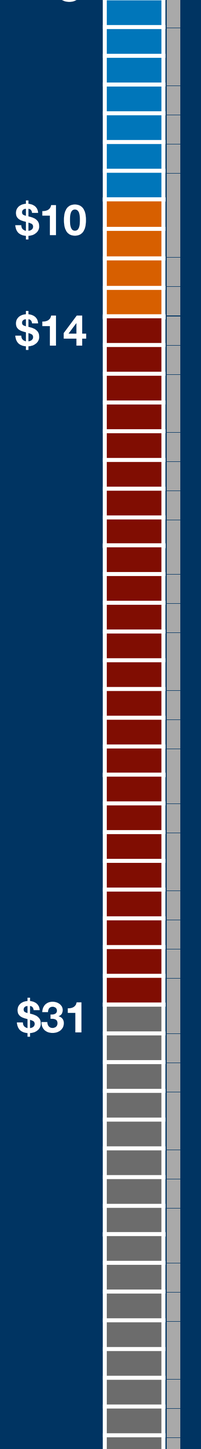

Int Off
cli
INHINT
0004
1
(disable int)

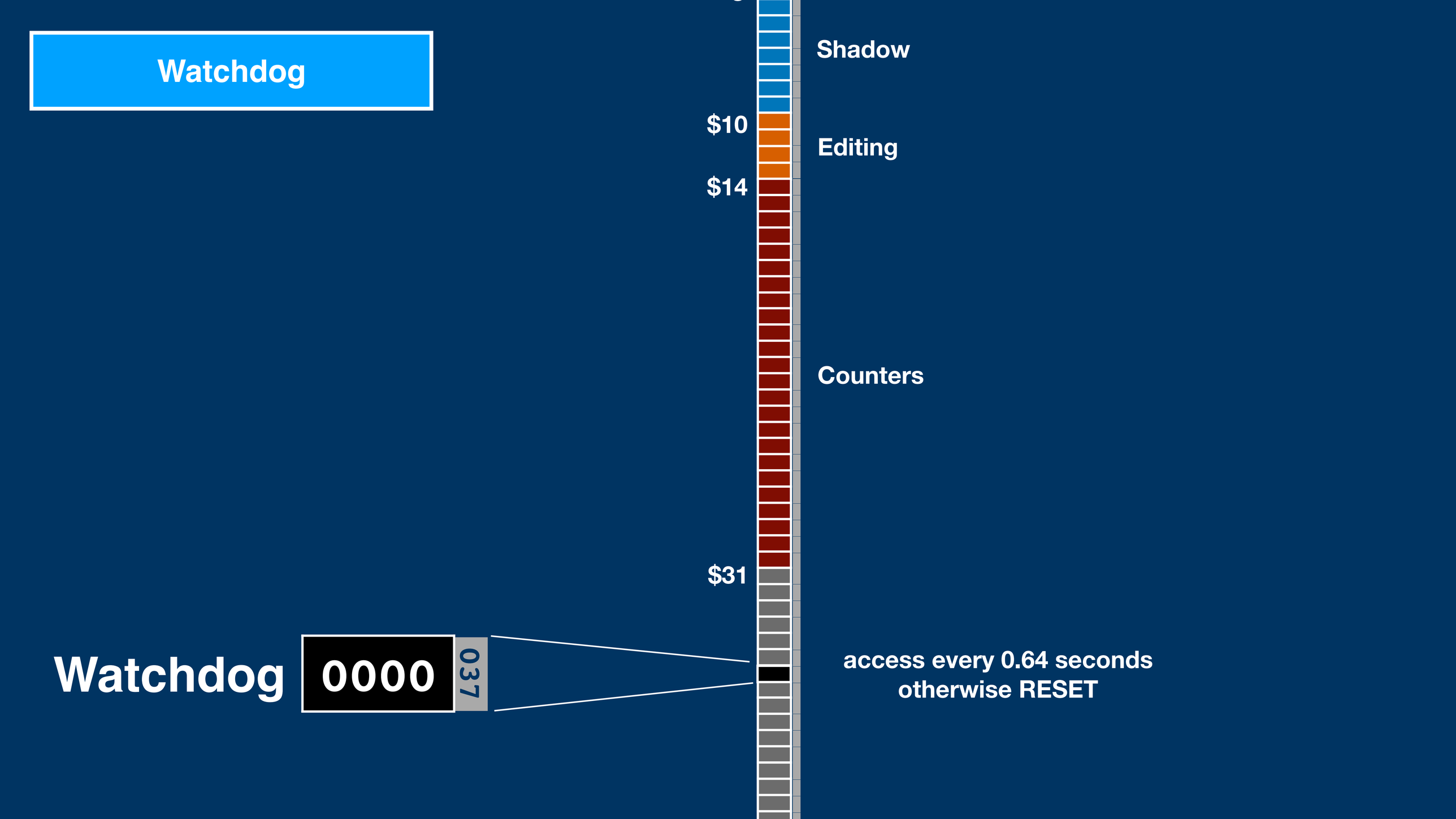

Int On
sti
RELINT
0003
1
(enable int)

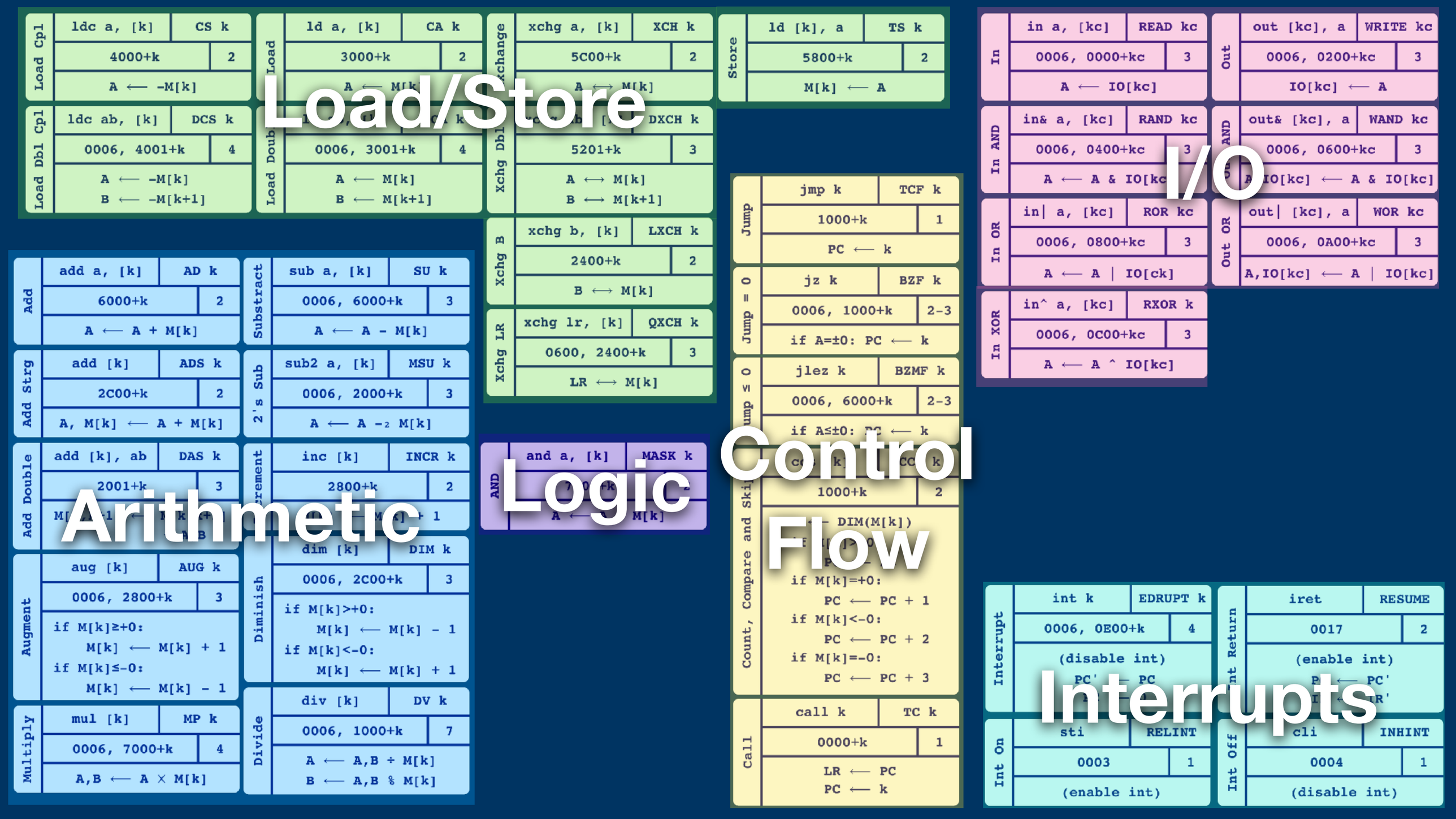
Watchdog



Watchdog

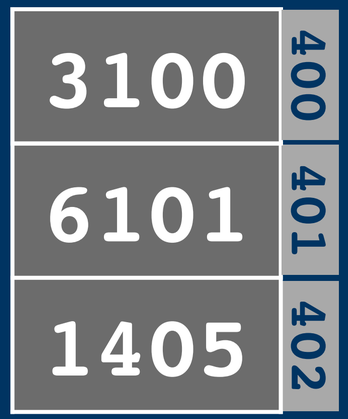



Watchdog
Shadow
$10
Editing
$14
Counters
$31
Watchdog
0000
037
access every 0.64 seconds
otherwise RESET


Load/Store
Load Cpl
ldc a, [k]
CS k
4000+k
2
A ⟵ -M[k]
Load Dbl Cpl
ldc ab, [k]
DCS k
0006, 4001+k
4
A ⟵ -M[k]
B ⟵ -M[k+1]
Load
ld a, [k]
CA k
3000+k
2
A ⟵ M[k]
Load Doub
0006, 3001+k
4
A ⟵ M[k]
B ⟵ M[k+1]
Exchange
xchg a, [k]
XCH k
5C00+k
2
A ⟷ M[k]
Xchg Dbl
DXCH k
5201+k
3
A ⟷ M[k]
B ⟷ M[k+1]
Xchg B
xchg b, [k]
LXCH k
2400+k
2
B ⟷ M[k]
Xchg LR
xchg lr, [k]
QXCH k
0600, 2400+k
3
LR ⟷ M[k]
Store
ld [k], a
TS k
5800+k
2
M[k] ⟵ A
I/O
In
in a, [kc]
READ kc
0006, 0000+kc
3
A ⟵ IO[kc]
Out
out [kc], a
WRITE kc
0006, 0200+kc
3
IO[kc] ⟵ A
In AND
in& a, [kc]
RAND kc
0006, 0400+kc
3
A ⟵ A & IO[kc]
Out AND
out& [kc], a
WAND kc
0006, 0600+kc
3
IO[kc] ⟵ A & IO[kc]
In OR
in| a, [kc]
ROR kc
0006, 0800+kc
3
A ⟵ A | IO[ck]
Out OR
out| [kc], a
WOR kc
0006, 0A00+kc
3
A,IO[kc] ⟵ A | IO[kc]
In XOR
in^ a, [kc]
RXOR k
0006, 0C00+kc
3
A ⟵ A ^ IO[kc]
Arithmetic
Add
add a, [k]
AD k
6000+k
2
A ⟵ A + M[k]
Substract
sub a, [k]
SU k
0006, 6000+k
3
A ⟵ A - M[k]
Add Strg
add [k]
ADS k
2C00+k
2
A, M[k] ⟵ A + M[k]
2's Sub
sub2 a, [k]
MSU k
0006, 2000+k
3
A ⟵ A -₂ M[k]
Add Double
add [k], ab
DAS k
2001+k
3
Increment
inc [k]
INCR k
2800+k
2
Augment
aug [k]
AUG k
0006, 2800+k
3
if M[k]≥+0:
M[k] ⟵ M[k] + 1
if M[k]≤-0:
M[k] ⟵ M[k] - 1
Diminish
dim [k]
DIM k
0006, 2C00+k
3
if M[k]>+0:
M[k] ⟵ M[k] - 1
if M[k]<-0:
M[k] ⟵ M[k] + 1
Multiply
mul [k]
MP k
0006, 7000+k
4
A,B ⟵ A × M[k]
Divide
div [k]
DV k
0006, 1000+k
7
A ⟵ A,B ÷ M[k]
B ⟵ A,B % M[k]
Logic
AND
and a, [k]
MASK k
A ⟵ A & M[k]
Control Flow
Jump
jmp k
TCF k
1000+k
1
PC ⟵ k
Jump = 0
jz k
BZF k
0006, 1000+k
2-3
if A=±0: PC ⟵ k
Jump ≤ 0
jlez k
BZMF k
0006, 6000+k
2-3
if A≤±0: PC ⟵ k
Count, Compare and Skip
1000+k
2
⟵ DIM(M[k])
if M[k]=+0:
PC ⟵ PC + 1
if M[k]<-0:
PC ⟵ PC + 2
if M[k]=-0:
PC ⟵ PC + 3
Call
call k
TC k
0000+k
1
LR ⟵ PC
PC ⟵ k
Interrupts
Interrupt
int k
EDRUPT k
0006, 0E00+k
4
(disable int)
Int Return
iret
RESUME
0017
2
(enable int)
Int On
sti
RELINT
0003
1
(enable int)
Int Off
cli
INHINT
0004
1
(disable int)

Instruction Encoding

Instruction Encoding




```
ld a, [$100]
add a, [$101]
jmp $405
```

Instruction Encoding

Instruction Encoding


```
ld a, [$100]
```

Instruction Encoding


```
ld a, [$100]
```

Instruction Encoding


```
ld a, [$100]
```

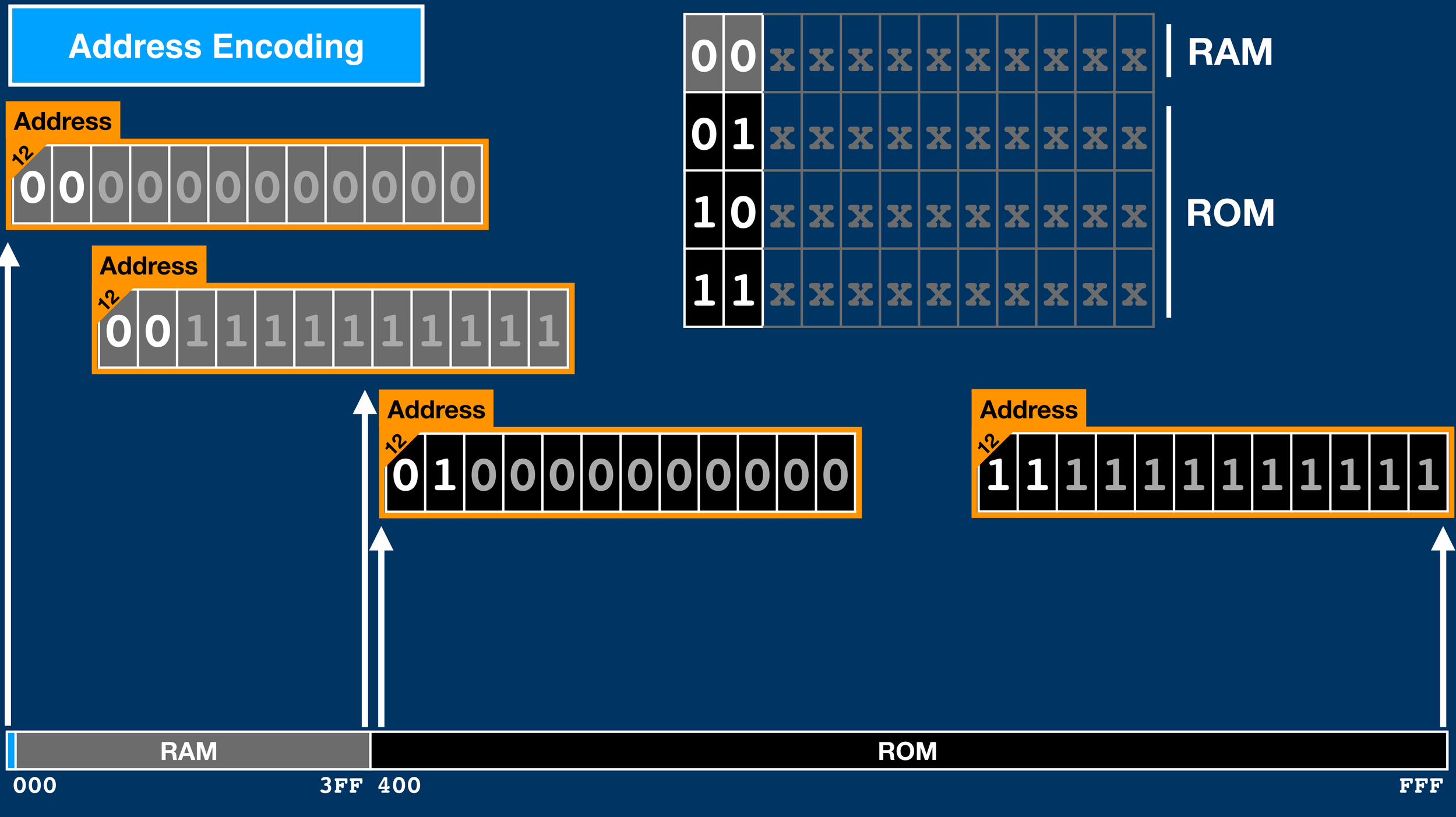
Instruction Encoding

Instruction Encoding

Opcode			Address											
0	0	0												call k
0	0	1												ccs [k]
0	1	0												inc [k]
0	1	1												ld a, [k]
1	0	0												ldc a, [k]
1	0	1												ld [k], a
1	1	0												add a, [k]
1	1	1												and a, [k]

Address Encoding

Address Encoding



Instruction Encoding

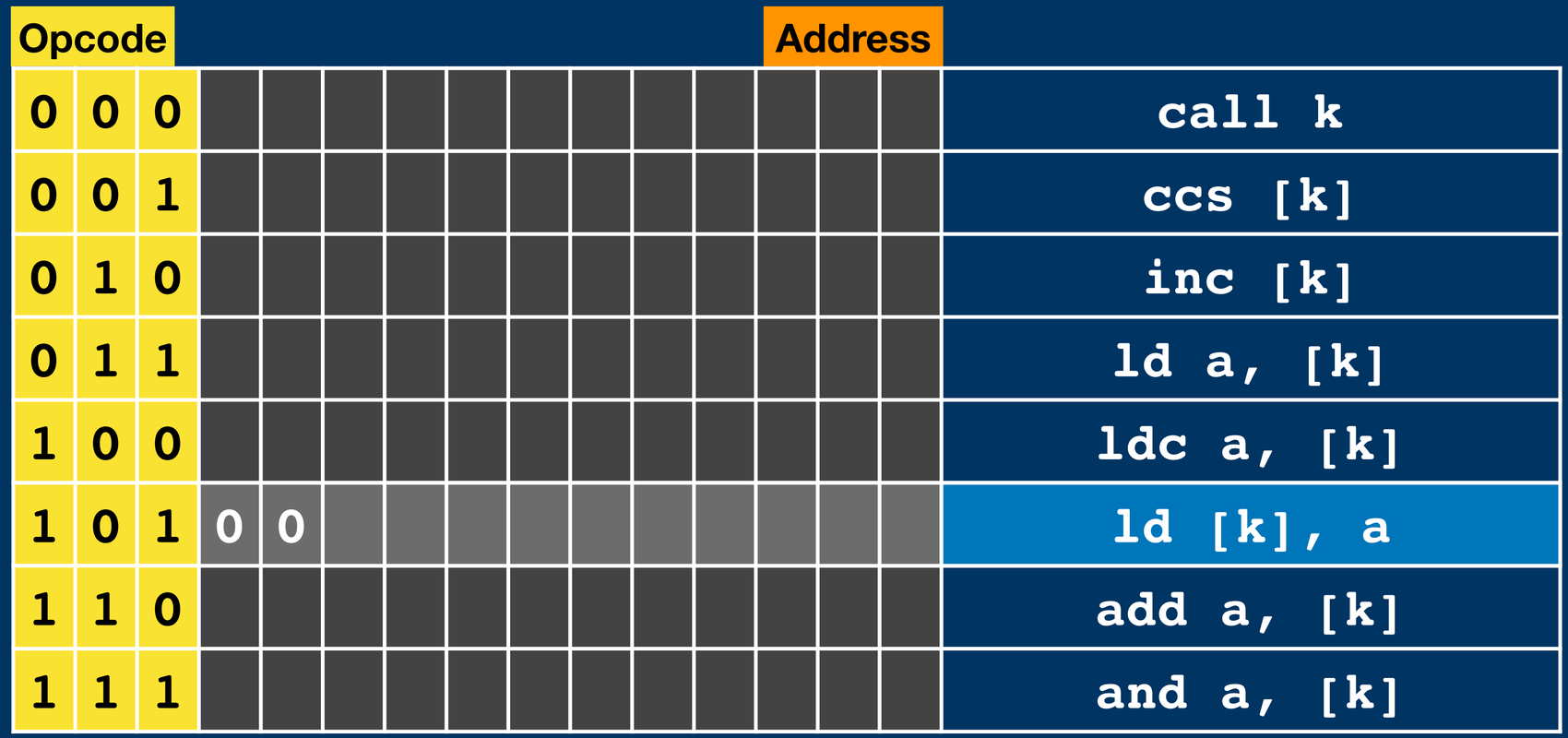
Opcode			Address											
0	0	0												call k
0	0	1												ccs [k]
0	1	0												inc [k]
0	1	1												ld a, [k]
1	0	0												ldc a, [k]
1	0	1												ld [k], a
1	1	0												add a, [k]
1	1	1												and a, [k]

RAM

RAM/ROM

ROM

Instruction Encoding

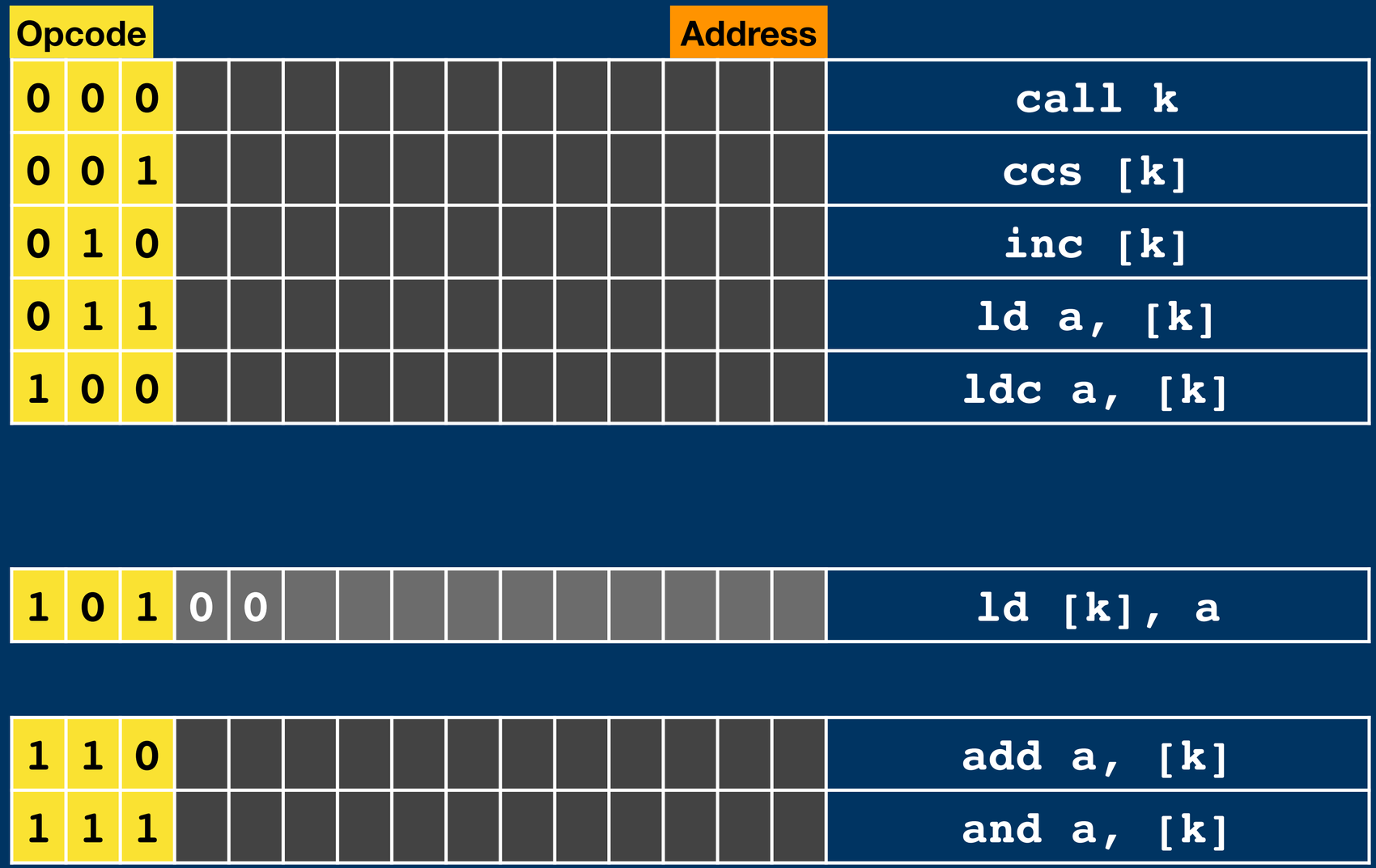


RAM

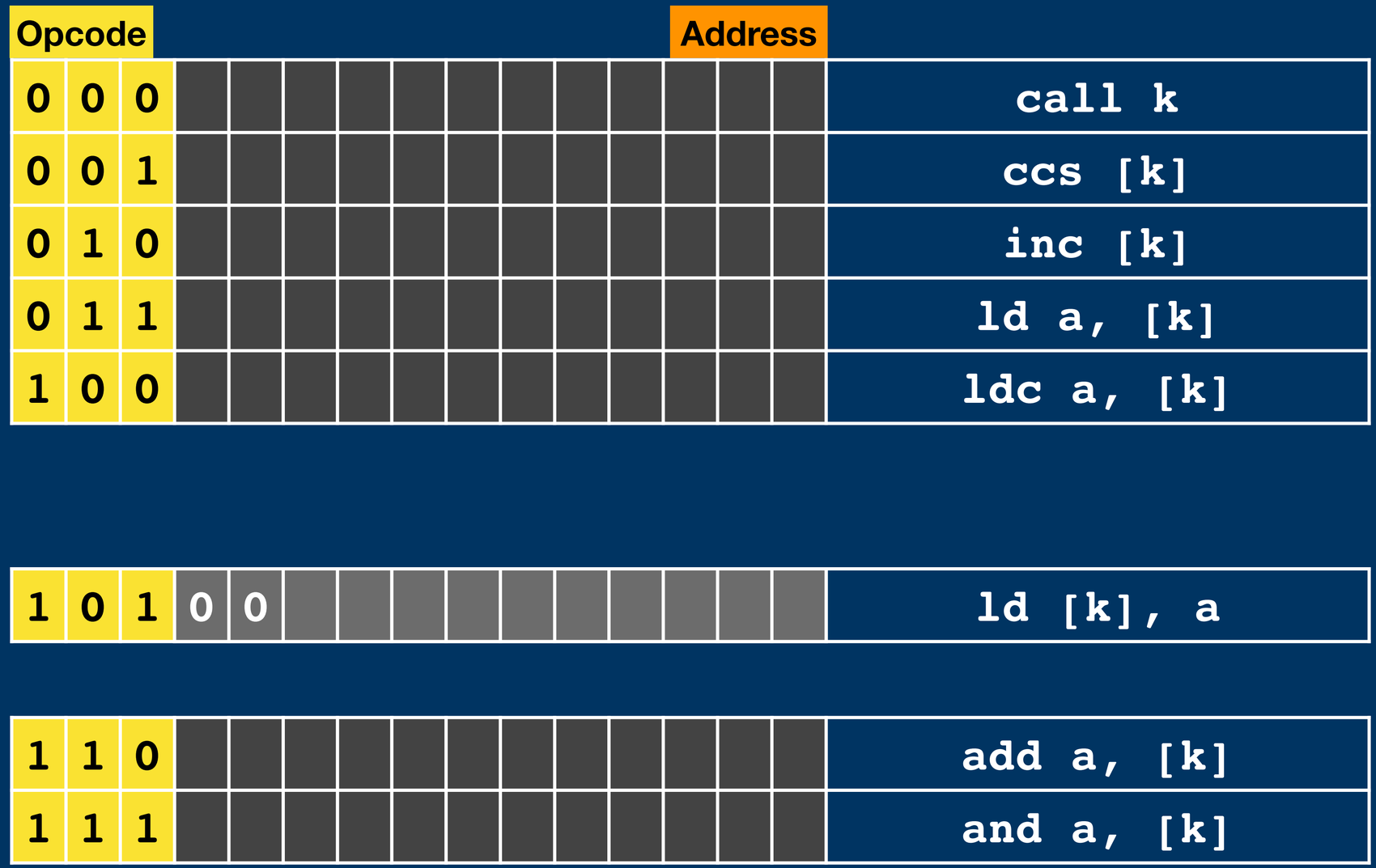
RAM/ROM

ROM

Instruction Encoding



Instruction Encoding



Instruction Encoding

Opcode			Address											
0	0	0												call k
0	0	1												ccs [k]
0	1	0												inc [k]
0	1	1												ld a, [k]
1	0	0												ldc a, [k]
1	0	1	0	0										INDEX
1	0	1	0	1										xchg ab, [k]
1	0	1	1	0										ld [k], a
1	0	1	1	1										xchg a, [k]
1	1	0												add a, [k]
1	1	1												and a, [k]

RAM

RAM/ROM

ROM

Instruction Encoding

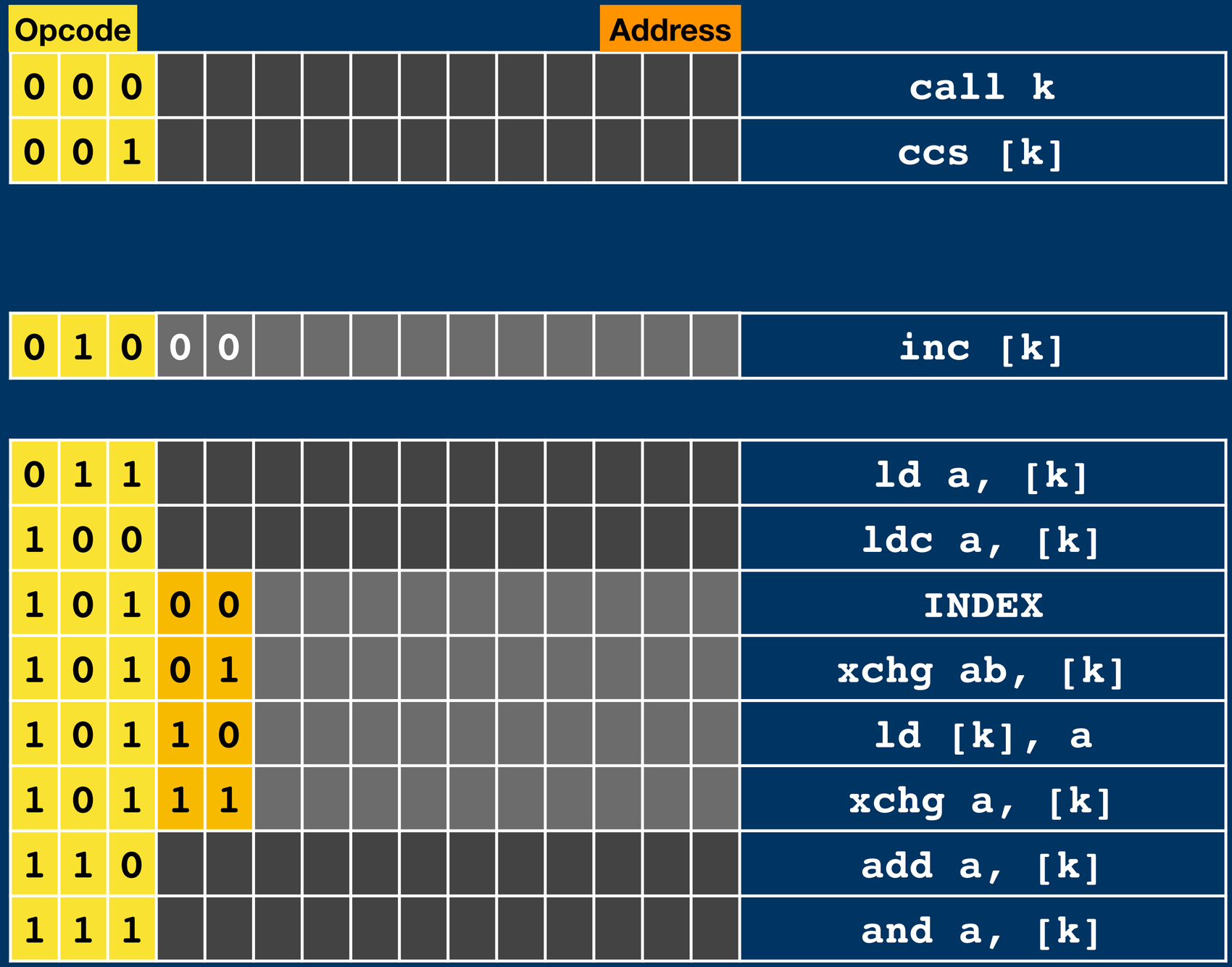
Opcode			Address											
0	0	0												call k
0	0	1												ccs [k]
0	1	0												inc [k]
0	1	1												ld a, [k]
1	0	0												ldc a, [k]
1	0	1	0	0										INDEX
1	0	1	0	1										xchg ab, [k]
1	0	1	1	0										ld [k], a
1	0	1	1	1										xchg a, [k]
1	1	0												add a, [k]
1	1	1												and a, [k]

RAM

RAM/ROM

ROM

Instruction Encoding

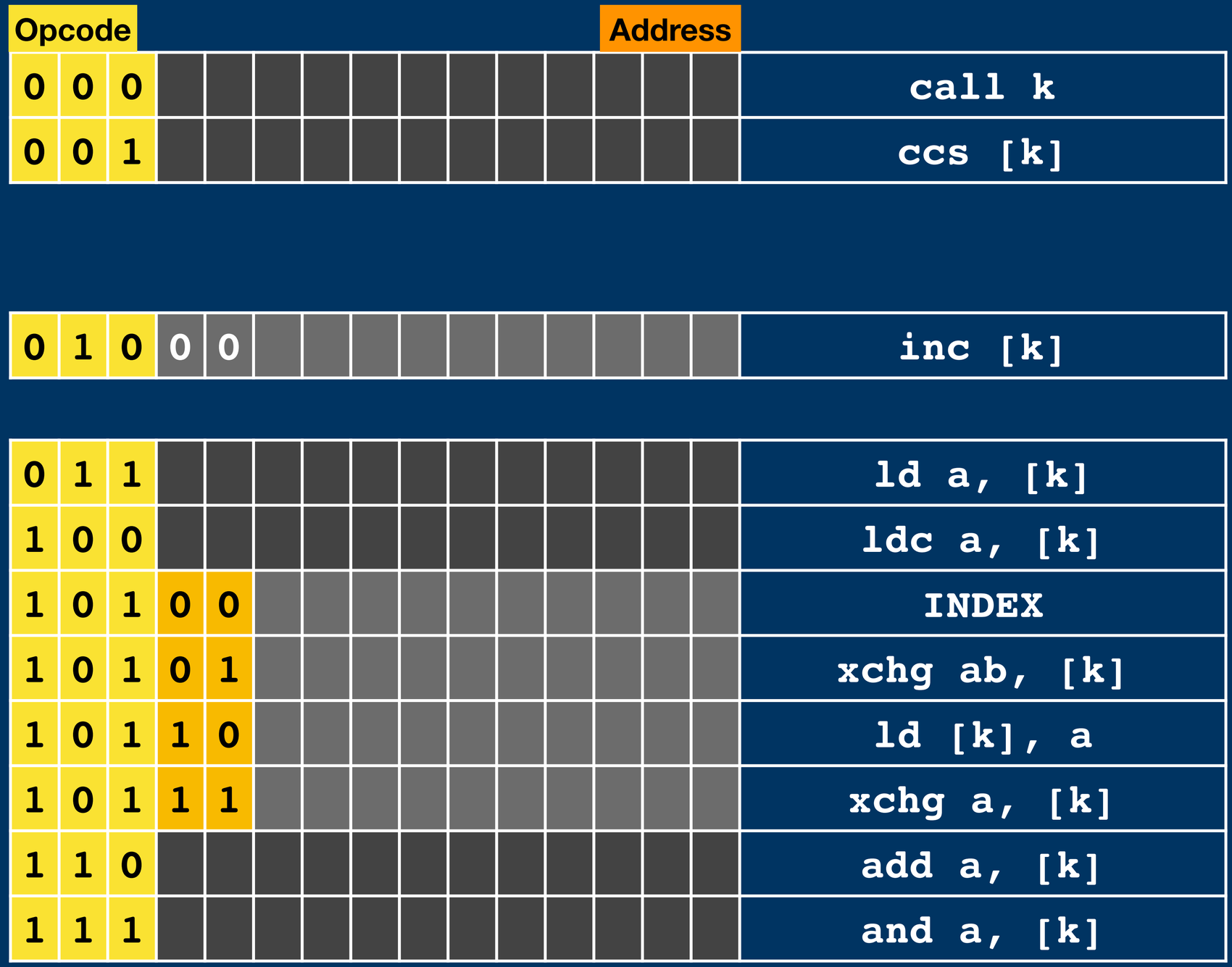


RAM

RAM/ROM

ROM

Instruction Encoding



RAM

RAM/ROM

ROM

Instruction Encoding

Opcode			Address											
0	0	0												call k
0	0	1												ccs [k]
0	1	0	0	0										add [k], ab
0	1	0	0	1										xchg b, [k]
0	1	0	1	0										inc [k]
0	1	0	1	1										add [k], a
0	1	1												ld a, [k]
1	0	0												ldc a, [k]
1	0	1	0	0										INDEX
1	0	1	0	1										xchg ab, [k]
1	0	1	1	0										ld [k], a
1	0	1	1	1										xchg a, [k]
1	1	0												add a, [k]
1	1	1												and a, [k]

RAM

RAM/ROM

ROM

Instruction Encoding

Opcode			Address											
0	0	0												call k
0	0	1												ccs [k]
0	1	0	0	0										add [k], ab
0	1	0	0	1										xchg b, [k]
0	1	0	1	0										inc [k]
0	1	0	1	1										add [k], a
0	1	1												ld a, [k]
1	0	0												ldc a, [k]
1	0	1	0	0										INDEX
1	0	1	0	1										xchg ab, [k]
1	0	1	1	0										ld [k], a
1	0	1	1	1										xchg a, [k]
1	1	0												add a, [k]
1	1	1												and a, [k]

RAM

RAM/ROM

ROM

Instruction Encoding

Opcode															Address
0	0	0													call k
0	0	1	0	0											ccs [k]

0	1	0	0	0											add [k], ab
0	1	0	0	1											xchg b, [k]
0	1	0	1	0											inc [k]
0	1	0	1	1											add [k], a
0	1	1													ld a, [k]
1	0	0													ldc a, [k]
1	0	1	0	0											INDEX
1	0	1	0	1											xchg ab, [k]
1	0	1	1	0											ld [k], a
1	0	1	1	1											xchg a, [k]
1	1	0													add a, [k]
1	1	1													and a, [k]

RAM

RAM/ROM

ROM

Instruction Encoding

Opcode	Address	
0 0 0		call k
0 0 1	0 0	ccs [k]

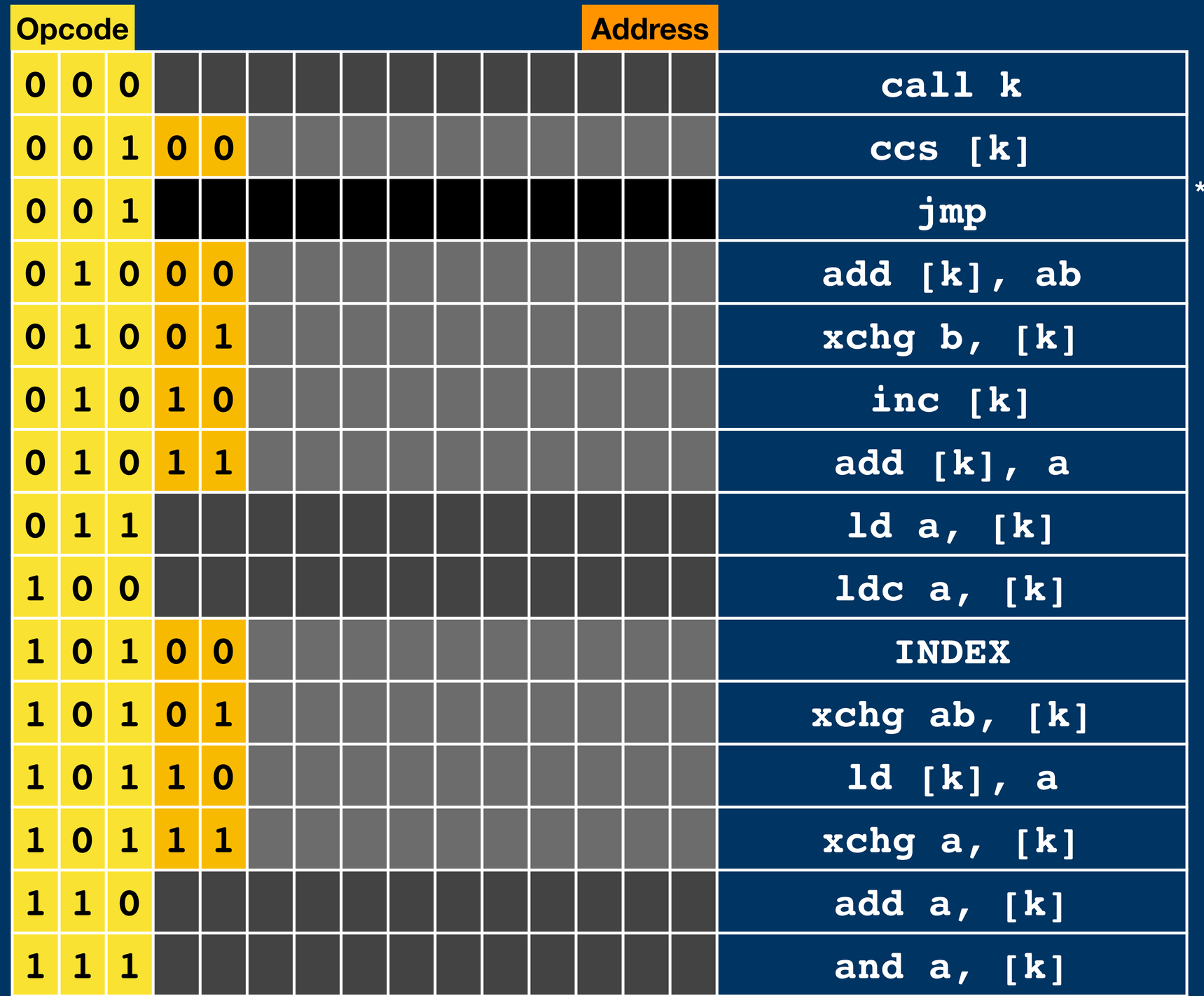
0 1 0 0 0	add [k], ab
0 1 0 0 1	xchg b, [k]
0 1 0 1 0	inc [k]
0 1 0 1 1	add [k], a
0 1 1	ld a, [k]
1 0 0	ldc a, [k]
1 0 1 0 0	INDEX
1 0 1 0 1	xchg ab, [k]
1 0 1 1 0	ld [k], a
1 0 1 1 1	xchg a, [k]
1 1 0	add a, [k]
1 1 1	and a, [k]

RAM

RAM/ROM

ROM

Instruction Encoding



* `call` can be used to work around the ROM limitation of `jmp`

RAM

RAM/ROM

ROM

Instruction Encoding

Instruction Encoding

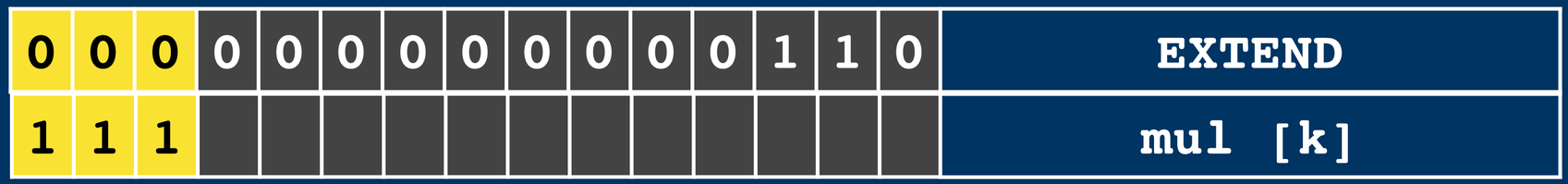
A	0007	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

Opcode			Address											
0	0	0												call k
0	0	0	0	0	0	0	0	0	0	0	0	1	1	sti
0	0	0	0	0	0	0	0	0	0	0	0	1	0	cli
0	0	0	0	0	0	0	0	0	0	0	0	1	1	EXTEND

Instruction Encoding

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	EXTEND
0	0	0													call k
0	0	1	0	0											ccs [k]
0	0	1													jmp
0	1	0	0	0											add [k], ab
0	1	0	0	1											xchg b, [k]
0	1	0	1	0											inc [k]
0	1	0	1	1											add [k], a
0	1	1													ld a, [k]
1	0	0													ldc a, [k]
1	0	1	0	0											INDEX
1	0	1	0	1											xchg ab, [k]
1	0	1	1	0											ld [k], a
1	0	1	1	1											xchg a, [k]
1	1	0													add a, [k]
1	1	1													and a, [k]

Instruction Encoding



Instruction Encoding

Instruction Encoding

EXTEND

I/O

RAM

RAM/ROM

ROM

Instruction Encoding

Opcode						Address									
0	0	0	0	0	0										in [kc]
0	0	0	0	0	1										out [kc]
0	0	0	0	1	0										in& [kc]
0	0	0	0	1	1										out& [kc]
0	0	0	1	0	0										in| [kc]
0	0	0	1	0	1										out| [kc]
0	0	0	1	1	0										in^ [kc]
0	0	0	1	1	1										edrupt k
0	0	1	0	0											div [k]
0	0	1													jz k
0	1	0	0	0											sub2 a, [k]
0	1	0	0	1											xchg lr, [k]
0	1	0	1	0											aug [k]
0	1	0	1	1											dim [k]
0	1	1													ld ab, [k]
1	0	0													ldc ab, [k]
1	0	1													INDEX
1	1	0	0	0											sub a, [k]
1	1	0													jlez k
1	1	1													mul [k]

EXTEND

I/O

RAM

RAM/ROM

ROM

Instruction Encoding

A	0007	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

Opcode			Address											
0	0	0												call k
0	0	0	0	0	0	0	0	0	0	0	0	1	1	sti
0	0	0	0	0	0	0	0	0	0	0	0	1	0	cli
0	0	0	0	0	0	0	0	0	0	0	0	1	1	EXTEND

Instruction Encoding

A	0007	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

Opcode			Address												
0	0	0													call k
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	ret
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	sti
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	cli
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	EXTEND

Return

ret

Return	ret	RETURN
	call 2 (alias)	1
	LR ← PC | PC ← 2	

A	0007	000
B	0F00	001
LR	0421	002
EB	0000	003
FB	0000	004
PC	0E81	005
BB	0000	006
0	0000	007

	3100	41F ld a, [$100]
	0E80	420 call $e80
	3103	421 ld a, [$103]
	5880	E80 ld [$80], a
PC →	0002	E81 ret

Return

ret

Return	ret	RETURN
	call 2 (alias)	1
	LR ← PC | PC ← 2	

A	0007	000
B	0F00	001
LR	0421	002
EB	0000	003
FB	0000	004
PC	0E81	005
BB	0000	006
0	0000	007

3100	41F	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]

	5880	E80	ld [$80], a
PC →	0002	E81	call 2

Return

ret

Return	ret	RETURN
	call 2 (alias)	1
	LR ← PC | PC ← 2	


```
ld a, [$100]
call $e80
ld a, [$103]

ld [$80], a
call 2
```

Return

ret

Return	ret	RETURN
	call 2 (alias)	1
	LR ← PC | PC ← 2	

A	0007	000
B	0F00	001
LR	0421	002
EB	0000	003
FB	0000	004
PC	0002	005
BB	0000	006
0	0000	007

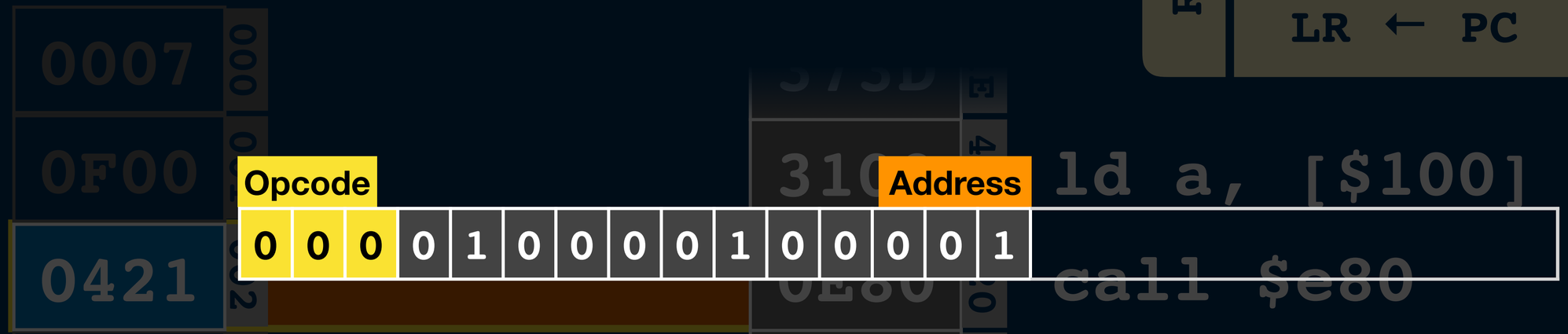
	3103	E40
	0E80	E41
	0421	E42
	5880	E80
	0002	E81

```
ld a, [$100]
call $e80
ld a, [$103]
ld [$80], a
call 2
```

Return

ret

Return	ret	RETURN
	call 2 (alias)	1
	LR ← PC | PC ← 2	



A	0007	
B	0F00	
LR	0421	
EB	0000	
FB	0000	
PC	0002	
BB	0000	
0	0000	

	3103	ld a, [$100]
	0E80	call $e80
	3103	ld a, [$103]
	5880	ld [$80], a
	0002	call 2

Return

ret

Return	ret	RETURN
	call 2 (alias)	1
	LR ← PC | PC ← 2	

A	0007	000
B	0F00	001
LR	0421	002
EB	0000	003
FB	0000	004
PC	0002	005
BB	0000	006
0	0000	007

3100	420	ld a, [$100]
0E80	420	call $e80
3103	421	ld a, [$103]
5880	E80	ld [$80], a
0002	E81	call 2

Return

ret

Return	ret	RETURN
	call 2 (alias)	1
	LR ← PC | PC ← 2	

A	0007	000
B	0F00	001
LR	0421	002
EB	0000	003
FB	0000	004
PC	0002	005
BB	0000	006
0	0000	007

3103	421	ld a, [$100]
		call $e80
3103	421	ld a, [$103]
5880	E80	ld [$80], a
0002	E81	call 2

Return

ret

Return	ret	RETURN
	call 2 (alias)	1
	LR ← PC | PC ← 2	

A	0007	000
B	0F00	001
LR	0421 *	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

* because it's a "call", LR will also be updated after every instruction (not shown here)

	3100	41F	ld a, [$100]
	0E80	420	call $e80
PC →	3103	421	ld a, [$103]
	5880	E80	ld [$80], a
	0002	E81	call 2

INDEX

INDEX

INDEX

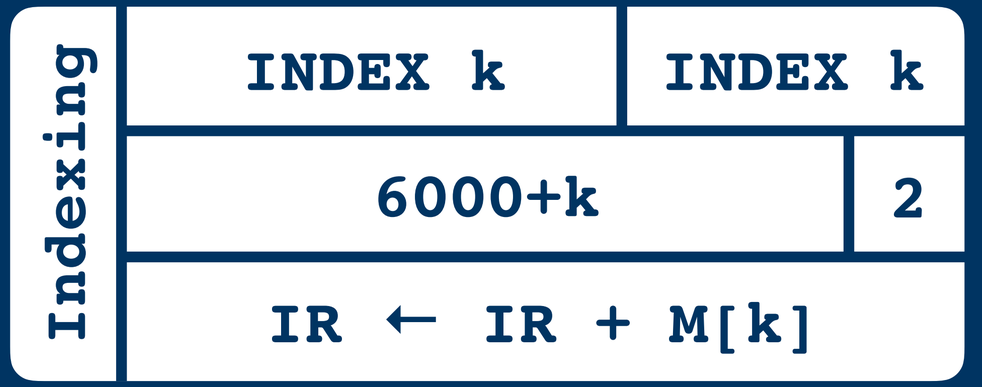

```
ld a, [$700+[$80]]
```

INDEX

1	0	1	0	0	0	0	1	0	0	0	0	0	0	INDEX [$80]
0	1	1	0	1	1	1	0	0	0	0	0	0	0	ld a, [$700]

```
ld a, [$700+[$80]]
```

INDEX



INDEX

Indexing	INDEX k	INDEX k
	6000+k	2
	IR ← IR + M[k]	

INDEX

Indexing	INDEX k	INDEX k
	6000+k	2
	IR ← IR + M[k]	

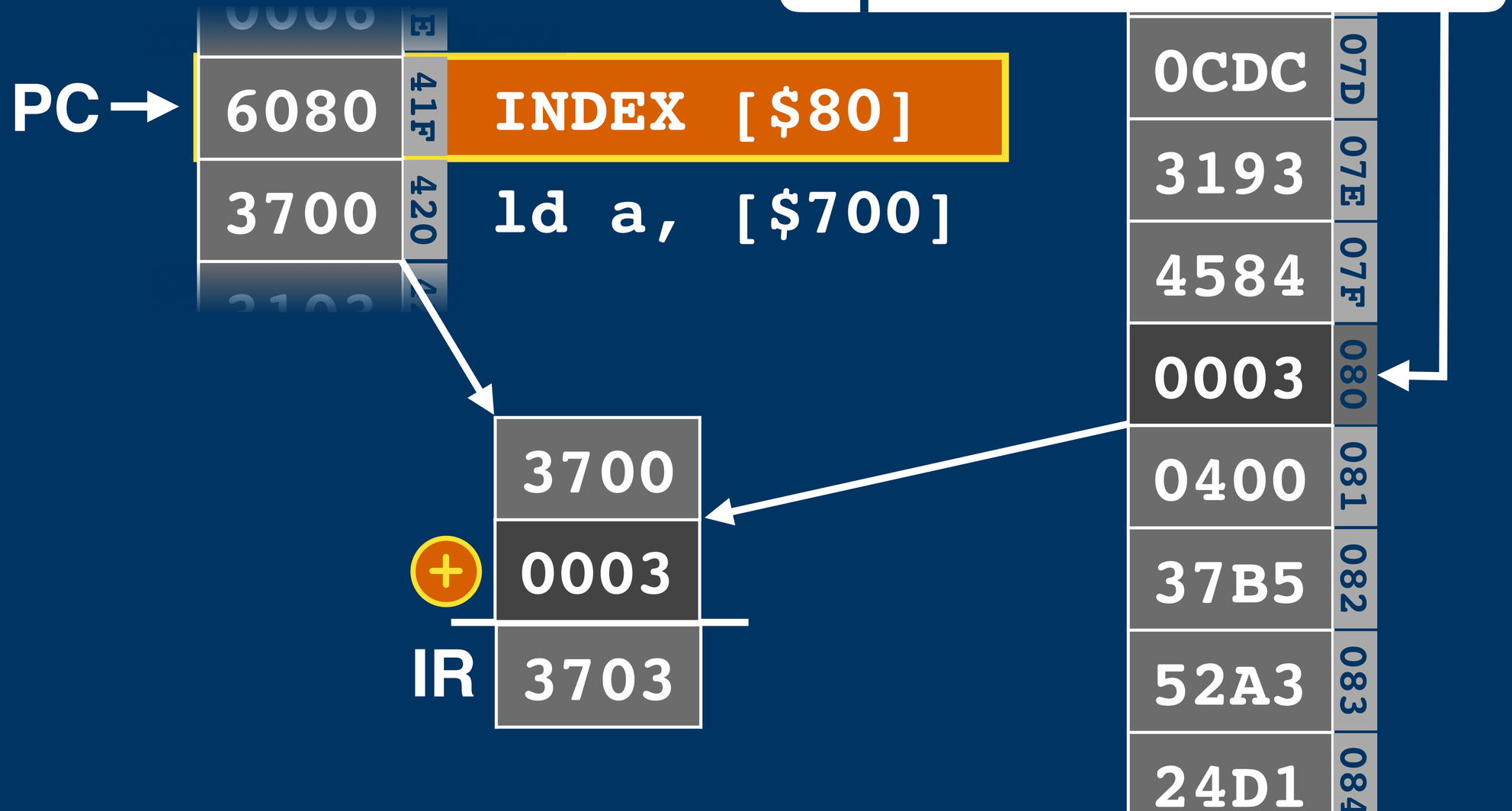
PC →

6080	41F	INDEX [$80]
3700	420	ld a, [$700]

0CDC	07D
3193	07E
4584	07F
0003	080
0400	081
37B5	082
52A3	083
24D1	084

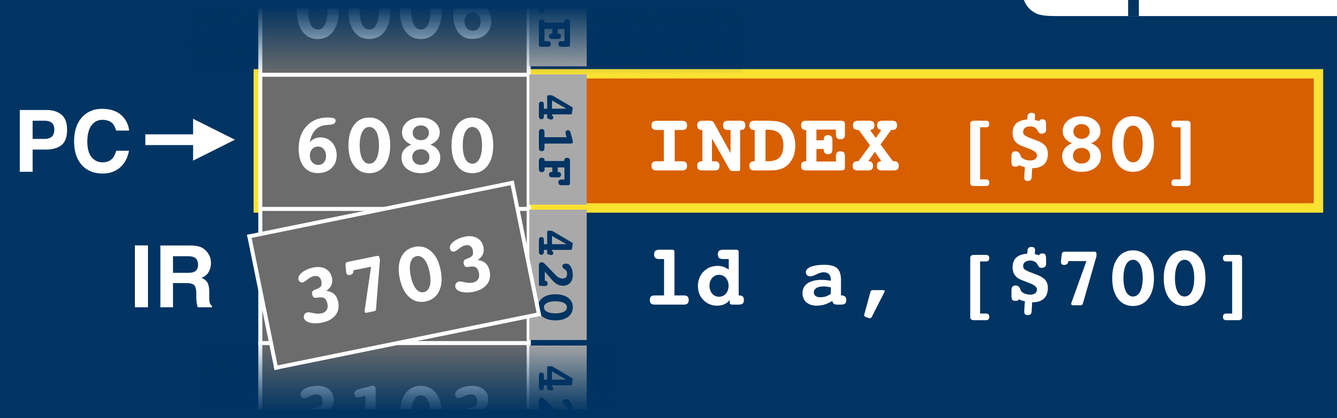
INDEX

Indexing	INDEX k	INDEX k
	6000+k	2
	IR ← IR + M[k]	

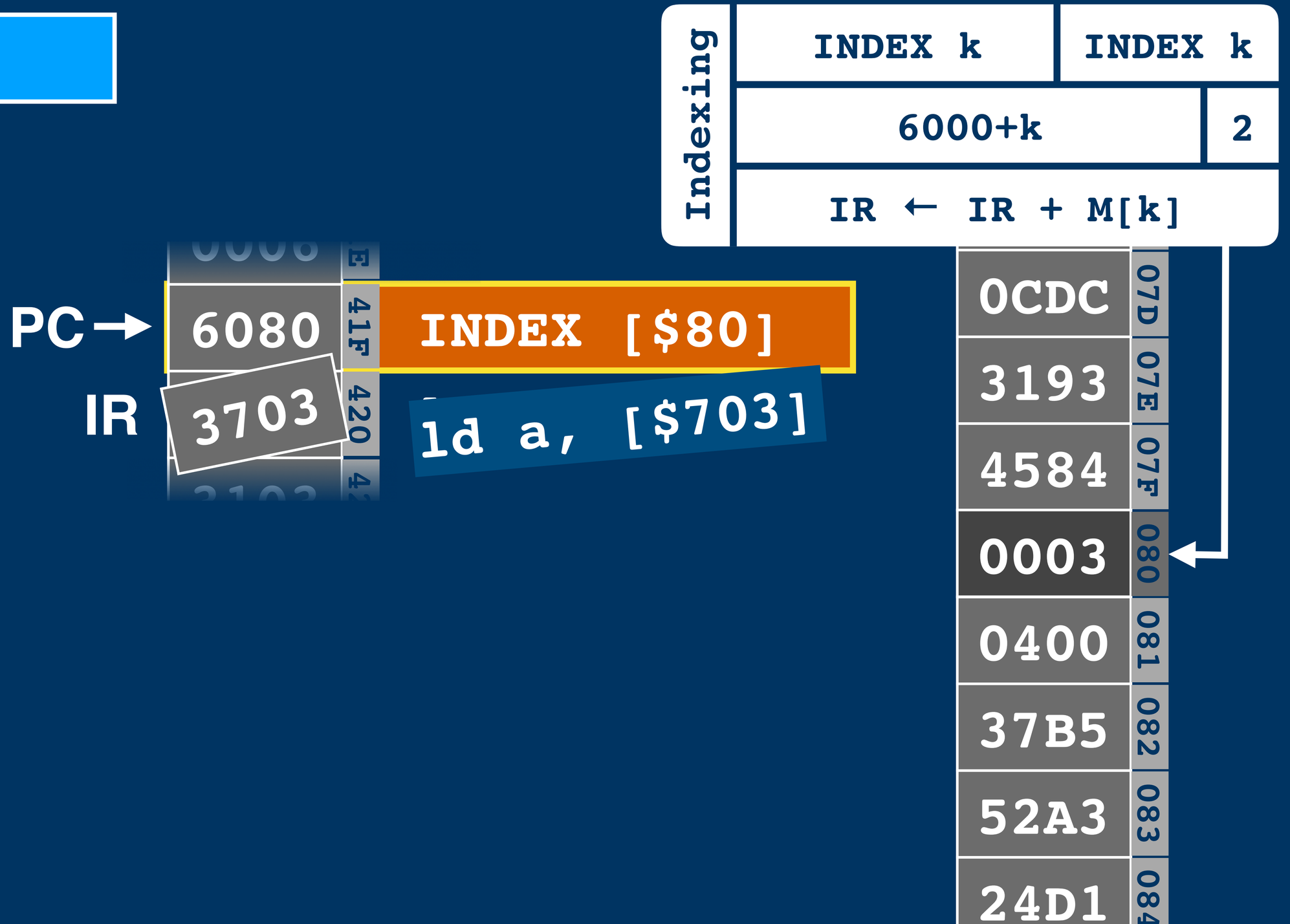


INDEX

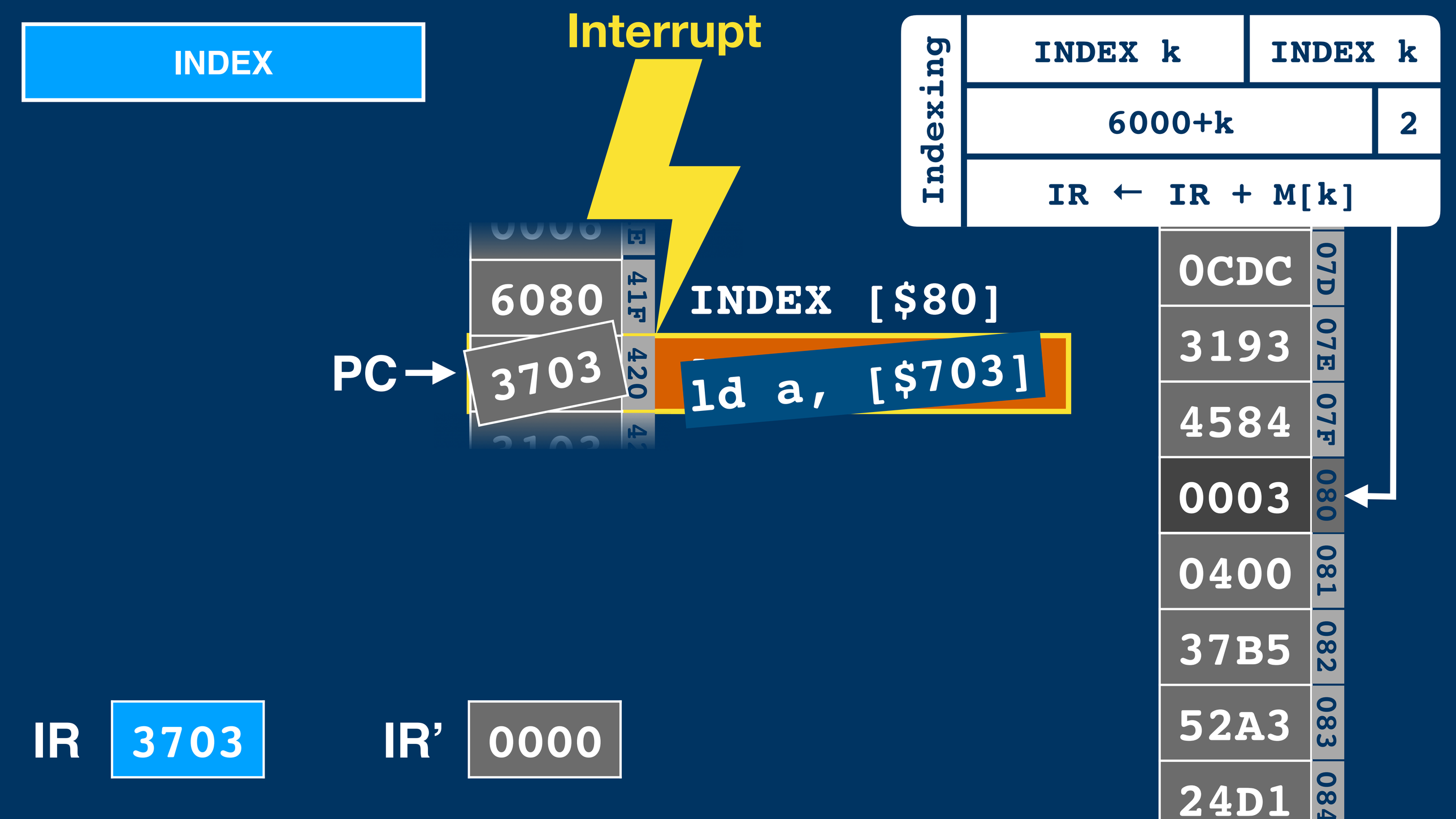
Indexing	INDEX k	INDEX k
	6000+k	2
	IR ← IR + M[k]	



INDEX

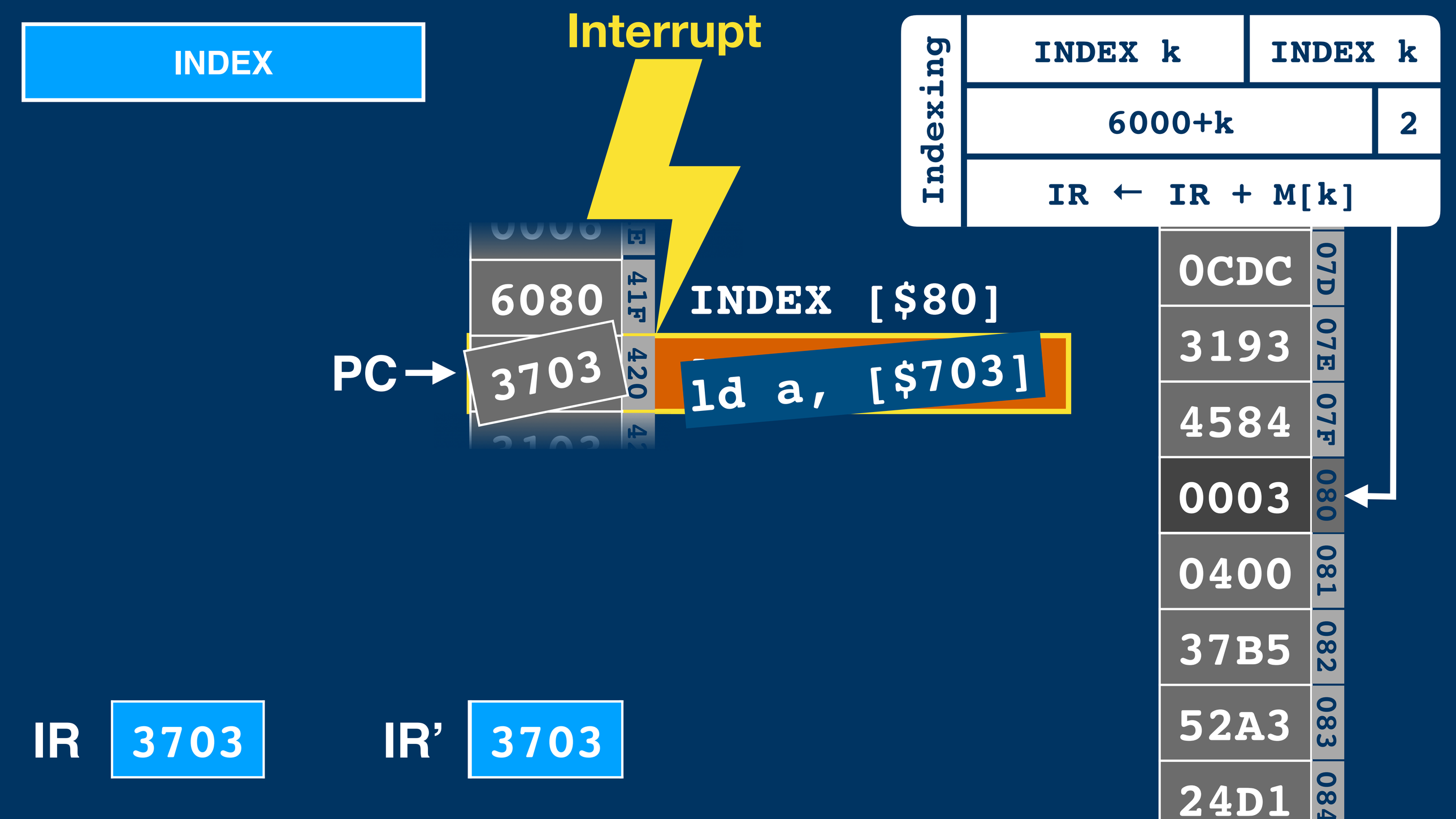

Indexing
INDEX k
INDEX k
6000+k
2
IR ← IR + M[k]
PC
6080
41F
INDEX [$80]
IR
3703
420
ld a, [$703]
07D
0CDC
07E
3193
07F
4584
080
0003
081
0400
082
37B5
083
52A3
084
24D1

INDEX



IR 3703

IR' 0000


INDEX
Interrupt
Indexing
INDEX k
INDEX k
6000+k
2
IR ← IR + M[k]
6080
41F
INDEX [$80]
PC
3703
420
ld a, [$703]
0CDC
07D
3193
07E
4584
07F
0003
080
0400
081
37B5
082
52A3
083
24D1
084
IR
3703
IR'
3703

Instruction Encoding

A’	0000	008
B’	0000	009
LR’	0000	00A
	0000	00B
	0000	00C
PC’	0000	00D
BB’	0000	00E
IR’	0000	00F

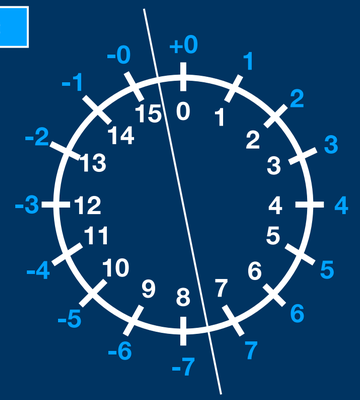
Instruction Encoding

A’	0000	008
B’	0000	009
LR’	0000	00A
	0000	00B
	0000	00C
PC’	0000	00D
BB’	0000	00E
IR’	0000	00F

Quirks

Quirks

One's Complement



Interrupts & Overflow

A	0000	V
B	0F00	
LR	05AA	
EB	0000	
FB	0000	
PC	0421	
BB	0000	
0	0000	

```
PC → ld a, [$200]
     add a, [$201]
     nop
     nop
     ld [$202], a
     jmp cont
     ld [$203], a
```

Interrupts disabled

Overflow Handling

A	0001
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

```
  0000
+ 0000   V
```

```
PC → ld a, [$200]
     add a, [$201]
     ld [$202], a
     jmp cont
     ld [$203], a
```

7FFF	
0001	
0000	Double Word Result
0001	

Conditional Jumps

Count, Compare and Skip

ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
    PC ← PC
if M[k]=+0:
    PC ← PC+1
if M[k]<-0:
    PC ← PC+2
if M[k]=-0:
    PC ← PC+3
```

1200	ccs [$200]	
0430	jmp $430	>+0
0440	jmp $440	=+0
0450	jmp $450	<-0
3080	ld a, [$80]	=-0

Editing

ROR 1 0 1 0 1 1 1 0 0 1 0 0 1 0 1

SHR 0 0 1 0 1 1 1 0 0 1 0 0 1 0 1

ROL 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0

SHR8 0 1 0 1 1 1 0 0 1 0 0 1 0 1 1

I/O Channels

B	
LR	

In	in a, [kc]	READ kc	
	0006, 0000+kc	2	
	A ← IO[kc]		
Out	out [kc], a	WRITE kc	
	0006, 0200+kc	2	
	IO[kc] ← A		
In AND	in& a, [kc]	RAND kc	
	0006, 0400+kc	2	
	A ← A & IO[kc]		
Out AND	out& [kc], a	WAND kc	
	0006, 0600+kc	2	
	A, IO[kc] ← A & IO[kc]		
In OR	in| a, [kc]	ROR kc	
	0006, 0800+kc	2	
	A ← A | IO[kc]		
Out OR	out| [kc], a	WOR kc	
	0006, 0A00+kc	2	
	A, IO[kc] ← A | IO[kc]		
In XOR	in^ a, [kc]	RXOR kc	
	0006, 0C00+kc	2	
	A ← A ^ IO[kc]		

INDEX

Interrupt

Indexing	INDEX k	INDEX k
	6000+k	2
	IR ← IR + M[k]	

```
PC → 6080  INDEX [$80]
     3700  ld a, [$702]
```

IR	3700
	0002
IR	3702

0CDC	
3193	
4584	
0002	
0400	
37B5	
52A3	
24D1	

Indexing

ld a, [$700+[$80]]

A	0006
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Table at $700

0000	
0000	
0000	
0003	
0006	
0009	
000C	
000F	
0012	

```
  0700
+ 0002
  0702
```

Index at $80

133A	
33DF	
1093	
23C9	
0002	
291C	
20F7	
100F	
45B3	

Quirks

One's Complement

Interrupts & Overflow

A	0000	V 000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

```
PC→ ld a, [$200]
    add a, [$201]
    nop
    nop
    ld [$202], a
    jmp cont
    ld [$203], a
```

Interrupts disabled

Overflow Handling

A	0001
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

```
  0000
  0000   V
```

```
PC→ ld a, [$200]
    add a, [$201]
    ld [$202], a
    jmp cont
    ld [$203], a
```

7FFF	
0001	
0000	Double Word Result
0001	

Conditional Jumps

ccs [k]	CCS k
1000+k	2

Count, Compare and Skip

```
A ← DABS(M[k])
if M[k]>+0:
    PC ← PC
if M[k]=+0:
    PC ← PC+1
if M[k]<-0:
    PC ← PC+2
if M[k]=-0:
    PC ← PC+3
```

1200	ccs [$200]	>+0
0430	jmp $430	=+0
0440	jmp $440	<-0
0450	jmp $450	=-0
3080	ld a, [$80]	

Editing

ROR 1 0 1 0 1 1 1 0 0 1 0 0 1 0 1

SHR 0 0 1 0 1 1 1 0 0 1 0 0 1 0 1

ROL 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0

SHR8 0 1 0 1 1 1 0 0 1 0 0 1 0 1 1

I/O Channels

B	000
LR	001
	002
	003
	004
	005
	006
	007

In	in a, [kc]	READ kc	
	0006, 0000+kc	2	
	A ← IO[kc]		
Out	out [kc], a	WRITE kc	
	0006, 0200+kc	2	
	IO[kc] ← A		
In AND	in& a, [kc]	RAND kc	
	0006, 0400+kc	2	
	A ← A & IO[kc]		
Out AND	out& [kc], a	WAND kc	
	0006, 0600+kc	2	
	A, IO[kc] ← A & IO[kc]		
In OR	in| a, [kc]	ROR kc	
	0006, 0800+kc	2	
	A ← A | IO[kc]		
Out OR	out| [kc], a	WOR kc	
	0006, 0A00+kc	2	
	A, IO[kc] ← A | IO[kc]		
In XOR	in^ a, [kc]	RXOR kc	
	0006, 0C00+kc	2	
	A ← A ^ IO[kc]		

INDEX

Interrupt

Indexing	INDEX k	INDEX k
	6000+k	2
	IR ← IR + M[k]	

```
PC→ 6080  INDEX [$80]
    3700  ld a, [$702]
```

IR	3700
	0002
IR	3702

0CDC	
3193	
4584	
0002	
0400	
37B5	
52A3	
24D1	

Indexing

ld a, [$700+[$80]]

A	0006
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Table at $700

0000	
0000	
0000	
0003	
0006	
0009	
000C	
000F	
0012	

```
  0700
  0002   +
  0702
```

Index at $80

133A	
33DF	
1093	
23C9	
0002	
291C	
20F7	
100F	
45B3	

Quirks

One's Complement

One's Complement

Interrupts & Overflow

Overflow Disables Interrupts

Overflow Handling

```
A  0001
B  0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0  0000
```

```
  0000
+ 0000
```

```
PC→ ld a, [$200]
    add a, [$201]
    ld [$202], a
    jmp cont
    ld [$203], a
```

7FFF	
0001	
0000	Double Word Result
0001	

Conditional Jumps

```
Count, Compare and Skip
ccs [k]   | CCS k
1000+k    | 2
A ← DABS(M[k])
if M[k]>+0:
    PC ← PC
if M[k]=+0:
    PC ← PC+1
if M[k]<-0:
    PC ← PC+2
if M[k]=-0:
    PC ← PC+3
```

```
1200  ccs [$200]
0430  jmp $430   >+0
0440  jmp $440   =+0
0450  jmp $450   <-0
3080  ld a, [$80]  =-0
```

Editing

```
ROR 1 0 1 0 1 1 1 0 0 1 0 0 1 0 1
SHR 0 0 1 0 1 1 1 0 0 1 0 0 1 0 1
ROL 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0
SHR8 0 1 0 1 1 1 0 0 1 0 0 1 0 1 1
```

I/O Channels

```
B
LR
```

In	in a, [kc]	READ kc	Out	out [kc], a	WRITE kc
	0006, 0000+kc	2		0006, 0200+kc	2
	A ← IO[kc]			IO[kc] ← A	
In AND	in& a, [kc]	RAND kc	Out AND	out& [kc], a	WAND kc
	0006, 0400+kc	2		0006, 0600+kc	2
	A ← A & IO[kc]			A, IO[kc] ← A & IO[kc]	
In OR	in| a, [kc]	ROR kc	Out OR	out| [kc], a	WOR kc
	0006, 0800+kc	2		0006, 0A00+kc	2
	A ← A | IO[kc]			A, IO[kc] ← A | IO[kc]	
In XOR	in^ a, [kc]	RXOR kc			
	0006, 0C00+kc	2			
	A ← A ^ IO[kc]				

INDEX

Interrupt

Indexing	INDEX k	INDEX k
	6000+k	2
	IR ← IR + M[k]	

```
PC→ 6080  INDEX [$80]
    3700  ld a, [$702]
```

```
IR  3700
  + 0002
IR  3702
```

```
0CDC
3193
4584
0002
0400
37B5
52A3
24D1
```

Indexing

```
ld a, [$700+[$80]]
```

```
A  0006
B  0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0  0000
```

Table at $700

```
  0700
+ 0002
  0702
```

Index at $80

0000	
0000	
0000	
0003	
0006	
0009	
000C	
000F	
0012	

133A	
33DF	
1093	
23C9	
0002	
291C	
20F7	
100F	
45B3	

Quirks

One's Complement

One's Complement

Interrupts & Overflow

Overflow Disables Interrupts

Overflow Handling

Store Skips On Overflow

Conditional Jumps

```
ccs [k]   | CCS k
1000+k    | 2
A ← DABS(M[k])
if M[k]>+0:
    PC ← PC
if M[k]=+0:
    PC ← PC+1
if M[k]<-0:
    PC ← PC+2
if M[k]=-0:
    PC ← PC+3
```

Count, Compare and Skip

```
1200  ccs [$200]
0430  jmp $430   >+0
0440  jmp $440   =+0
0450  jmp $450   <-0
3080  ld a, [$80]  =-0
```

Editing

ROR 1 0 1 0 1 1 1 0 0 1 0 0 1 0 1

SHR 0 0 1 0 1 1 1 0 0 1 0 0 1 0 1

ROL 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0

SHR8 0 1 0 1 1 1 0 0 1 0 0 1 0 1 1

I/O Channels

In	in a, [kc]	READ kc	
	0006, 0000+kc	2	
	A ← IO[kc]		
Out	out [kc], a	WRITE kc	
	0006, 0200+kc	2	
	IO[kc] ← A		
In AND	in& a, [kc]	RAND kc	
	0006, 0400+kc	2	
	A ← A & IO[kc]		
Out AND	out& [kc], a	WAND kc	
	0006, 0600+kc	2	
	A, IO[kc] ← A & IO[kc]		
In OR	in| a, [kc]	ROR kc	
	0006, 0800+kc	2	
	A ← A | IO[kc]		
Out OR	out| [kc], a	WOR kc	
	0006, 0A00+kc	2	
	A, IO[kc] ← A | IO[kc]		
In XOR	in^ a, [kc]	RXOR kc	
	0006, 0C00+kc	2	
	A ← A ^ IO[kc]		

INDEX

Interrupt

Indexing	INDEX k	INDEX k
	6000+k	2
	IR ← IR + M[k]	

PC → 6080 INDEX [$80]

3700 ld a, [$702]

IR 3700

\+ 0002

IR 3702

Indexing

ld a, [$700+[$80]]

A	0006
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Table at $700

\+ 0700 + 0002 = 0702

Index at $80

0000	
0000	
0000	
0003	
0006	
0009	
000C	
000F	
0012	

133A	
33DF	
1093	
23C9	
0002	
291C	
20F7	
100F	
45B3	

Quirks

One's Complement

One's Complement

Interrupts & Overflow

A	0000	000
B	0F00	001
LR	05AA	002
EB	0000	003
FB	0000	004
PC	0421	005
BB	0000	006
0	0000	007

Overflow Disables Interrupts

```
add a, [$201]
nop
ld [$202], a
ld [$203], a
```

Interrupts disabled

Overflow Handling

A	0001
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Store Skips On Overflow

```
PC→ ld a, [$200]
    add a, [$201]
    ld [$202], a
    jmp cont
    ld [$203], a
```

0000
0000

7FFF	200	Double Word Result
0001	201	
0001	202	
	203	

Conditional Jumps

Count, Compare and Skip

ccs [k]	CCS k
1000+k	2

```
A ← DABS(M[k])
if M[k]>+0:
    PC ← PC
if M[k]=+0:
    PC ← PC+1
if M[k]<-0:
    PC ← PC+2
if M[k]=-0:
    PC ← PC+3
```

CCS

```
1200  ccs [$200]
      jmp $430   >+0
      jmp $440   =+0
      jmp $450   <-0
0450  jmp $450   =-0
3080  ld a, [$80]
```

Editing

ROR 1 0 1 0 1 1 1 0 0 1 0 0 1 0 1

SHR 0 0 1 0 1 1 1 0 0 1 0 0 1 0 1

ROL 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0

SHR8 0 1 0 1 1 1 0 0 1 0 0 1 0 1 1

I/O Channels

B	000
LR	001
	002
	003
	004
	005
	006
	007

In	in a, [kc]	READ kc	Out	out [kc], a	WRITE kc
	0006, 0000+kc	2		0006, 0200+kc	2
	A ← IO[kc]			IO[kc] ← A	
In AND	in& a, [kc]	RAND kc	Out AND	out& [kc], a	WAND kc
	0006, 0400+kc	2		0006, 0600+kc	2
	A ← A & IO[kc]			A, IO[kc] ← A & IO[kc]	
In OR	in| a, [kc]	ROR kc	Out OR	out| [kc], a	WOR kc
	0006, 0800+kc	2		0006, 0A00+kc	2
	A ← A | IO[kc]			A, IO[kc] ← A | IO[kc]	
In XOR	in^ a, [kc]	RXOR kc			
	0006, 0C00+kc	2			
	A ← A ^ IO[kc]				

INDEX

Interrupt

Indexing	INDEX k	INDEX k
	6000+k	2
	IR ← IR + M[k]	

PC→	6080	41F	INDEX [$80]
	3700	420	ld a, [$702]

IR 3700
⊕ 0002
IR 3702

0CDC	07D	
3193	07E	
4584	07F	
0002	080	
0400	081	
37B5	082	
52A3	083	
24D1	084	

Indexing

ld a, [$700+[$80]]

A	0006
B	0F00
LR	05AA
EB	0000
FB	0000
PC	0421
BB	0000
0	0000

Table at $700

0000	6FE	
0000	6FF	
0000	700	
0003	701	
0006	702	
0009	703	
000C	704	
000F	705	
0012	706	

⊕ 0700
0002
0702

Index at $80

133A	07C	
33DF	07D	
1093	07E	
23C9	07F	
0002	080	
291C	081	
20F7	082	
100F	083	
45B3	084	

Quirks

One's Complement

One's Complement

Interrupts & Overflow

Overflow Disables Interrupts

Overflow Handling

Store Skips On Overflow

Conditional Jumps

CCS

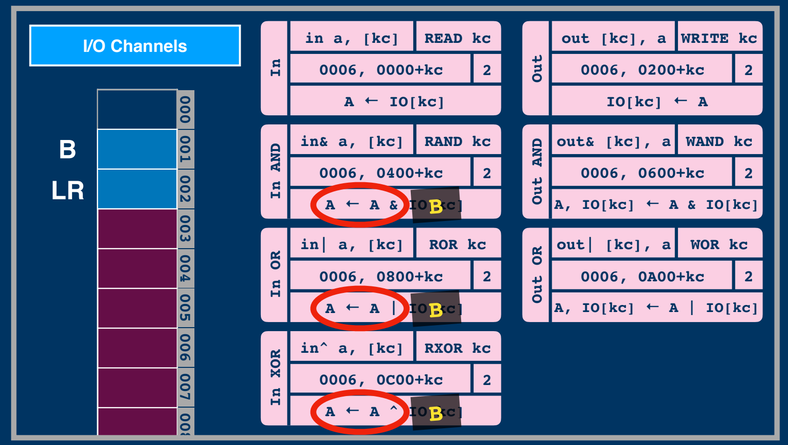
Editing

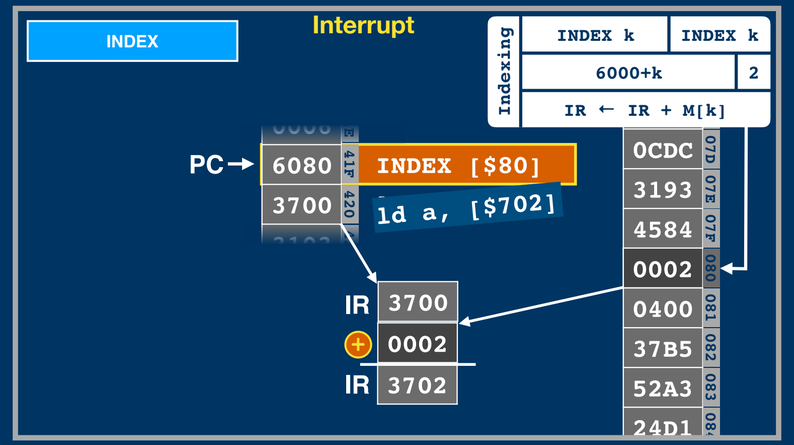
Shift/Rotate Using Memory

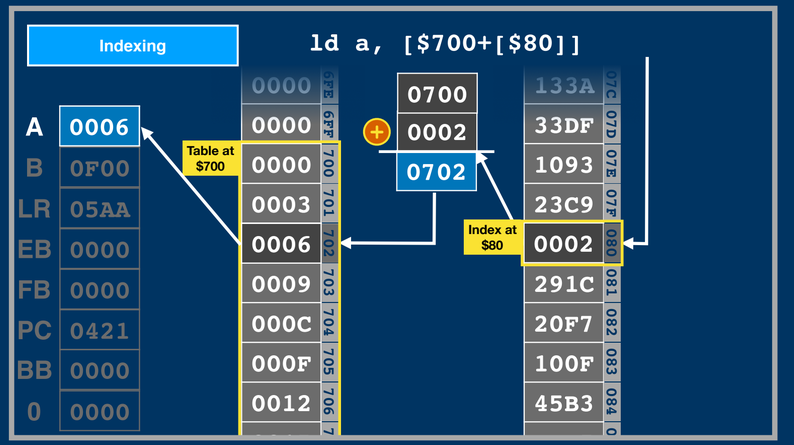
I/O Channels

INDEX

Indexing

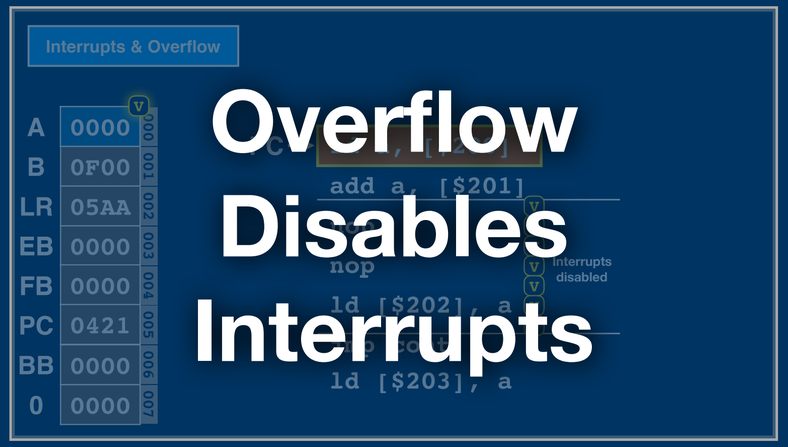




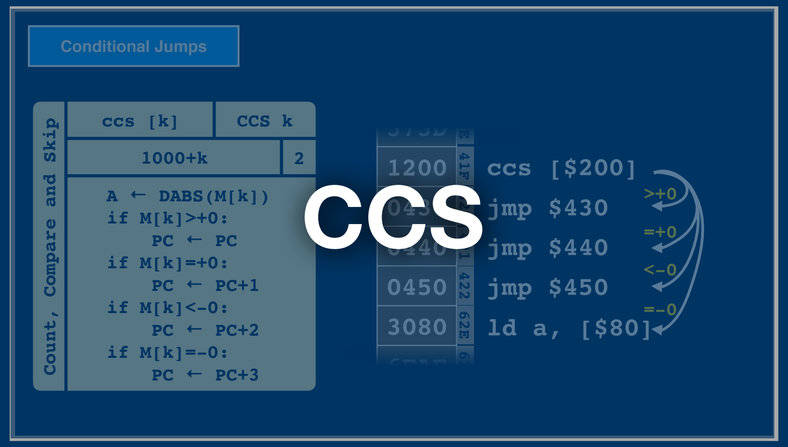


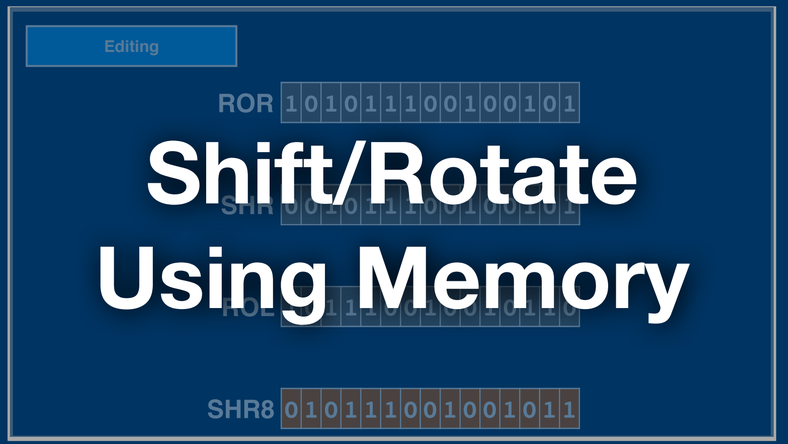
Quirks

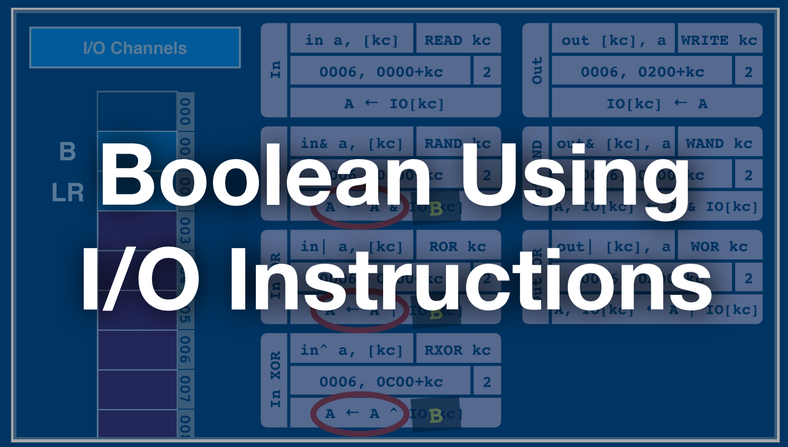

One's Complement
One's Complement

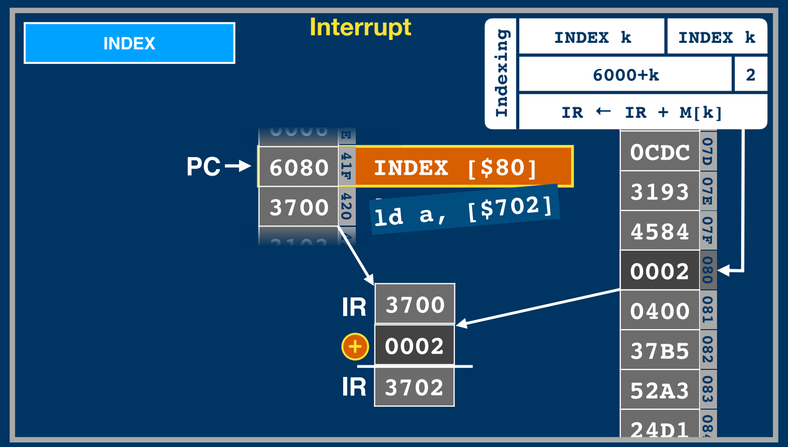

Interrupts & Overflow
A 0000
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
add a, [$201]
nop
ld [$202], a
ld [$203], a
Interrupts disabled
Overflow Disables Interrupts

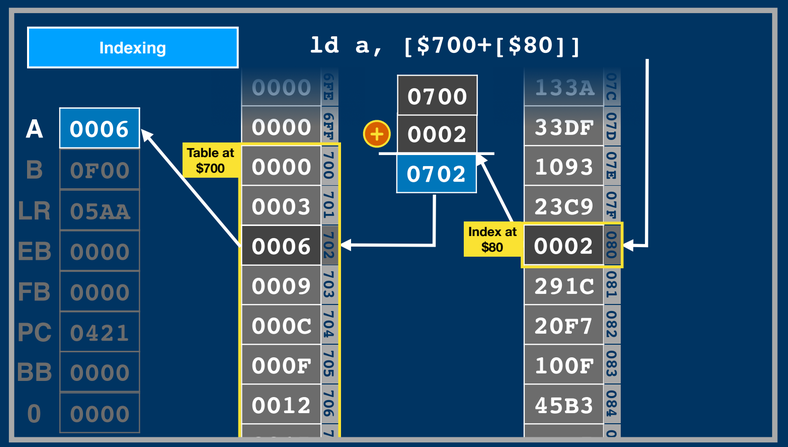

Overflow Handling
A 0001
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
0000
7FFF
0001
PC → ld a, [$200]
ld [$202], a
jmp cont
ld [$203], a
Double Word Result
Store Skips On Overflow


Conditional Jumps
Count, Compare and Skip
ccs [k] CCS k
1000+k 2
A ← DABS(M[k])
if M[k]>+0:
PC ← PC
if M[k]=+0:
PC ← PC+1
if M[k]<-0:
PC ← PC+2
if M[k]=-0:
PC ← PC+3
1200 ccs [$200]
jmp $430
jmp $440
0450 jmp $450
3080 ld a, [$80]
>+0
=+0
<-0
=-0
CCS


Editing
ROR 1010111001001011
SHR 0101110010010111
ROL 1011100100101110
SHR8 0101110010010111
Shift/Rotate Using Memory

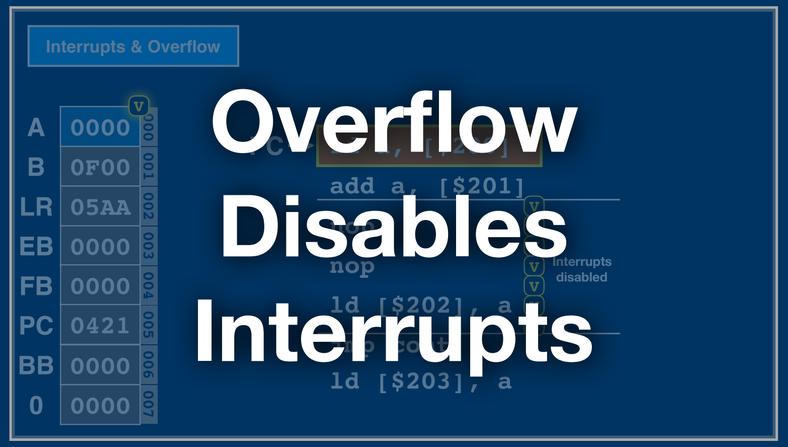

I/O Channels
B
LR
in a, [kc] READ kc
0006, 0000+kc 2
A ← IO[kc]
out [kc], a WRITE kc
0006, 0200+kc 2
IO[kc] ← A
in& a, [kc] RAND kc
out& [kc], a WAND kc
in| a, [kc] ROR kc
out| [kc], a WOR kc
in^ a, [kc] RXOR kc
0006, 0C00+kc 2
A ← A ^ IO[kc]
Boolean Using I/O Instructions


INDEX
Interrupt
Indexing
INDEX k INDEX k
6000+k 2
IR ← IR + M[k]
PC → 6080 INDEX [$80]
3700 ld a, [$702]
IR 3700
0002
IR 3702
0CDC
3193
4584
0002
0400
37B5
52A3
24D1

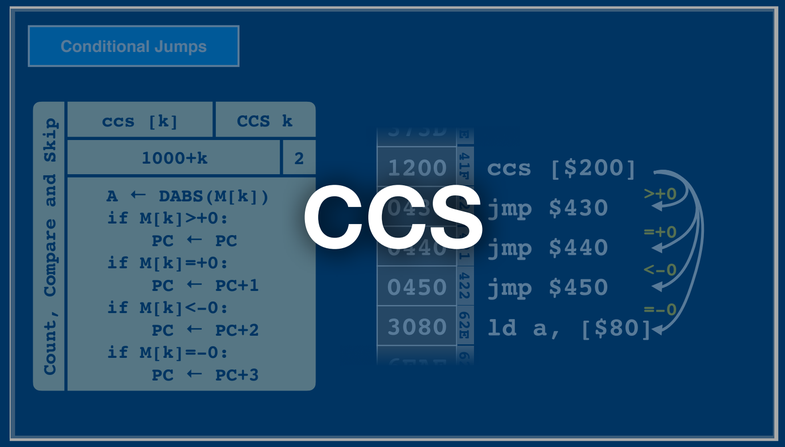

Indexing
ld a, [$700+[$80]]
A 0006
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
Table at $700
Index at $80
0700
0002
0702
0000
0000
0003
0006
0009
000C
000F
0012
133A
33DF
1093
23C9
0002
291C
20F7
100F
45B3

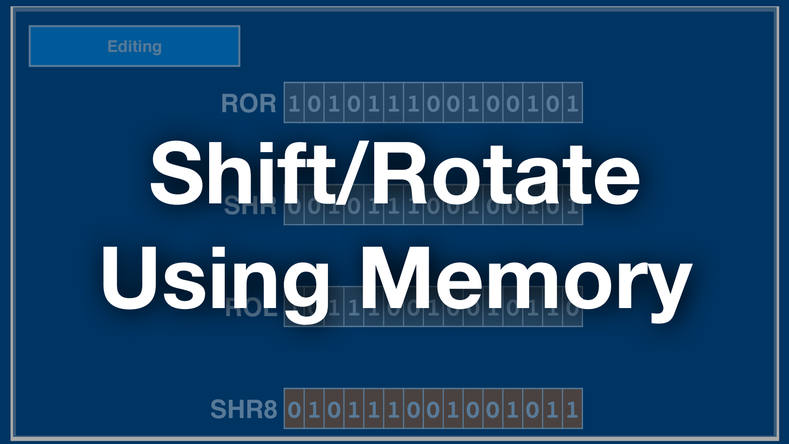
Quirks

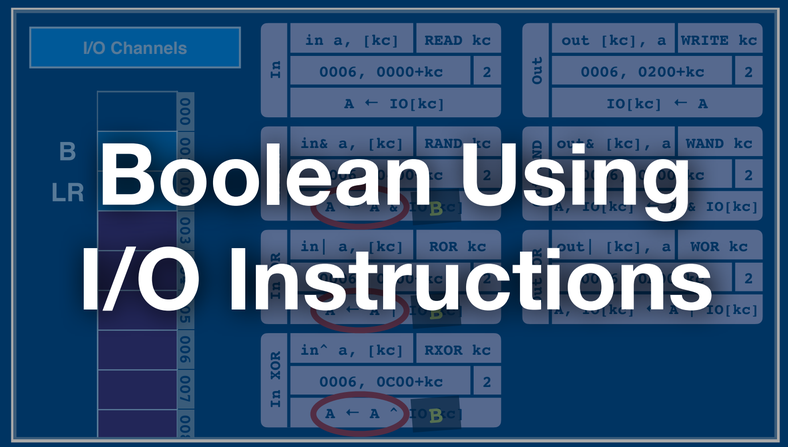

One's Complement
One's Complement

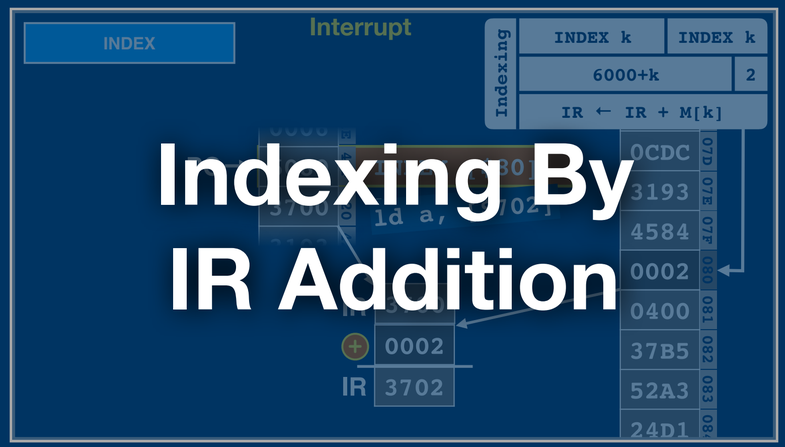

Interrupts & Overflow
A 0000
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
add a, [$201]
nop
ld [$202], a
ld [$203], a
Interrupts disabled
Overflow Disables Interrupts

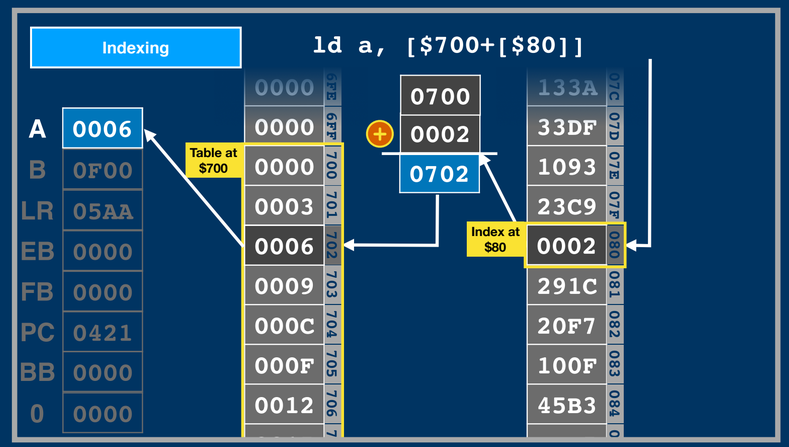

Overflow Handling
A 0001
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
0000
PC → ld a, [$200]
ld [$202], a
jmp cont
ld [$203], a
7FFF
0001
0001
Double Word Result
Store Skips On Overflow


Conditional Jumps
Count, Compare and Skip
ccs [k]
CCS k
1000+k
2
A ← DABS(M[k])
if M[k]>+0:
PC ← PC
if M[k]=+0:
PC ← PC+1
if M[k]<-0:
PC ← PC+2
if M[k]=-0:
PC ← PC+3
1200 ccs [$200]
jmp $430
jmp $440
0450 jmp $450
3080 ld a, [$80]
>+0
=+0
<-0
=-0
CCS


Editing
ROR 1010111001001011
SHR 0101110010010111
ROL 1011100100101110
SHR8 0101110010010111
Shift/Rotate Using Memory

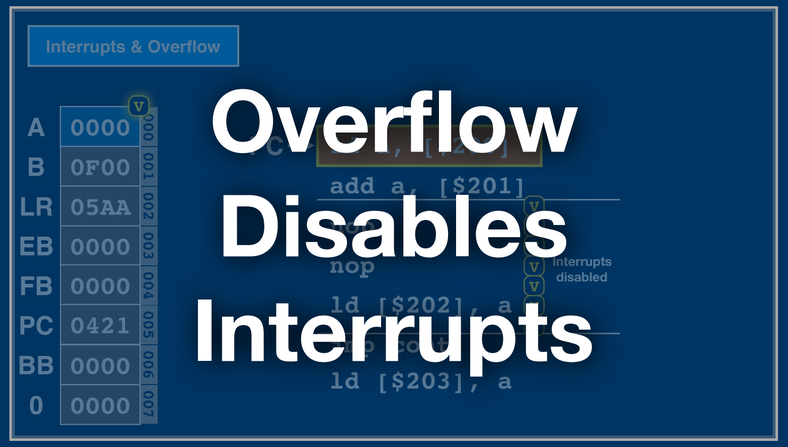

I/O Channels
B
LR
in a, [kc] READ kc
0006, 0000+kc 2
A ← IO[kc]
out [kc], a WRITE kc
0006, 0200+kc 2
IO[kc] ← A
in& a, [kc] RAND kc
out& [kc], a WAND kc
in| a, [kc] ROR kc
out| [kc], a WOR kc
in^ a, [kc] RXOR kc
0006, 0C00+kc 2
A ← A ^ IO[kc]
Boolean Using I/O Instructions


INDEX
Interrupt
Indexing
INDEX k
INDEX k
6000+k
2
IR ← IR + M[k]
ld a, [$700+[$80]]
0002
IR 3702
0CDC
3193
4584
0002
0400
37B5
52A3
Indexing By IR Addition

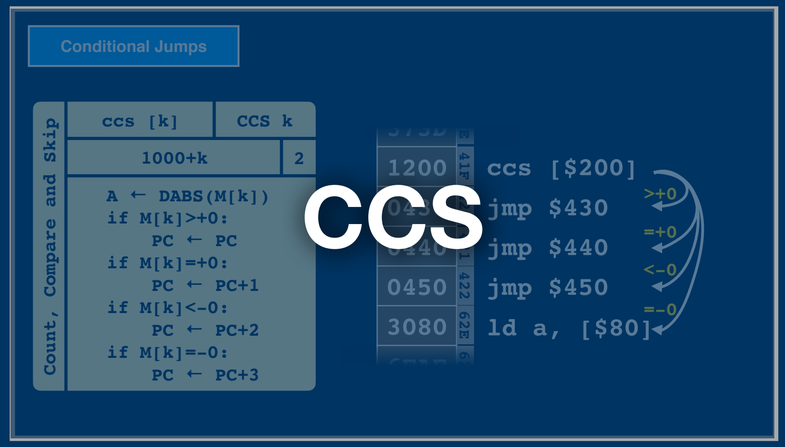

Indexing
ld a, [$700+[$80]]
A 0006
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
Table at $700
Index at $80
0700
0002
0702
0000
0000
0000
0003
0006
0009
000C
000F
0012
133A
33DF
1093
23C9
0002
291C
20F7
100F
45B3

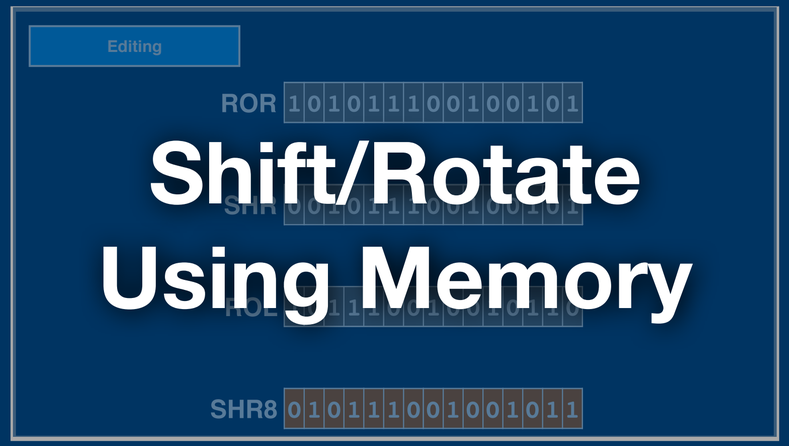
Quirks

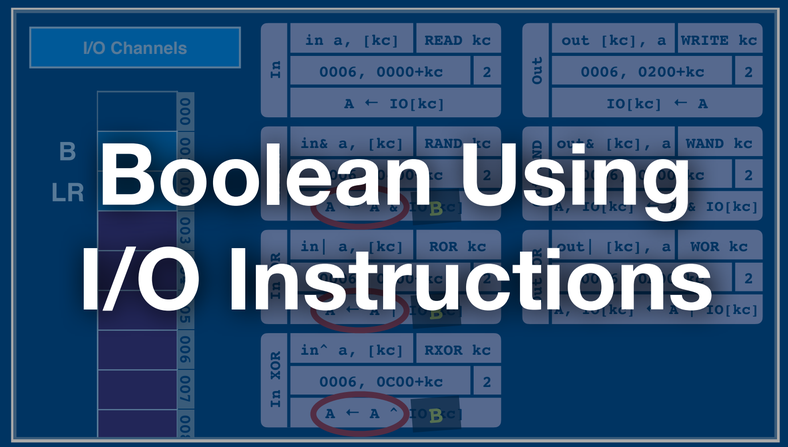

One's Complement
One's Complement

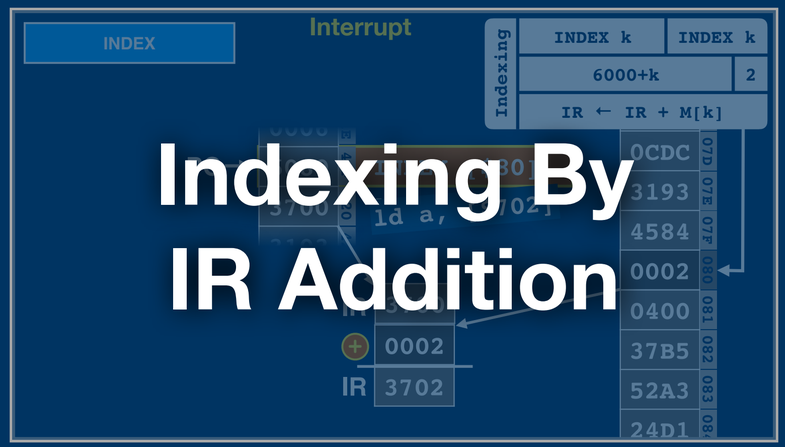

Interrupts & Overflow
A 0000
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
add a, [$201]
nop
ld [$202], a
ld [$203], a
Interrupts disabled
Overflow Disables Interrupts

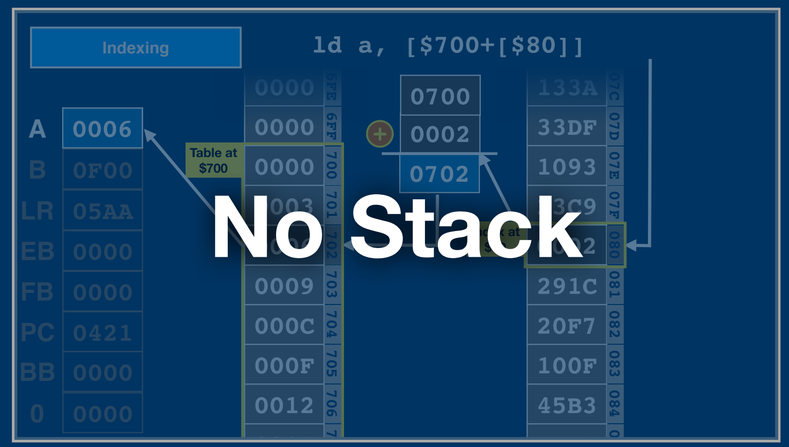

Overflow Handling
A 0001
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
0000
PC → ld a, [$200]
ld [$202], a
jmp cont
ld [$203], a
7FFF
0001
0001
Double Word Result
Store Skips On Overflow


Conditional Jumps
Count, Compare and Skip
ccs [k] CCS k
1000+k 2
A ← DABS(M[k])
if M[k]>+0:
PC ← PC
if M[k]=+0:
PC ← PC+1
if M[k]<-0:
PC ← PC+2
if M[k]=-0:
PC ← PC+3
1200 ccs [$200]
jmp $430
jmp $440
0450 jmp $450
3080 ld a, [$80]
>+0
=+0
<-0
=-0
CCS


Editing
ROR 1010111001001011
SHR 0101110010010111
ROL 1101001001011101
SHR8 0101110010010111
Shift/Rotate Using Memory


I/O Channels
B
LR
In
in a, [kc] READ kc
0006, 0000+kc 2
A ← IO[kc]
Out
out [kc], a WRITE kc
0006, 0200+kc 2
IO[kc] ← A
in& a, [kc] RAND kc
out& [kc], a WAND kc
in| a, [kc] ROR kc
out| [kc], a WOR kc
In XOR
in^ a, [kc] RXOR kc
0006, 0C00+kc 2
A ← A ^ IO[kc]
Boolean Using I/O Instructions


INDEX
Interrupt
Indexing
INDEX k INDEX k
6000+k 2
IR ← IR + M[k]
ld a, [$3700+[$02]]
0002
IR 3702
0CDC
3193
4584
0002
0400
37B5
52A3
24D1
Indexing By IR Addition


Indexing
ld a, [$700+[$80]]
A 0006
B 0F00
LR 05AA
EB 0000
FB 0000
PC 0421
BB 0000
0 0000
Table at $700
0700
0002
0702
0000
0000
0003
0009
000C
000F
0012
133A
33DF
1093
291C
20F7
100F
45B3
No Stack

Architecture

- Instruction Set
- Memory Model
- Arithmetic
- I/O
- Interrupts
- Encoding
- Counters

Architecture

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

Hardware

Hardware

Hardware

1 MHz

Microcode

Integrated Circuits

Core Memory

Core Rope Memory

Hardware

Hardware

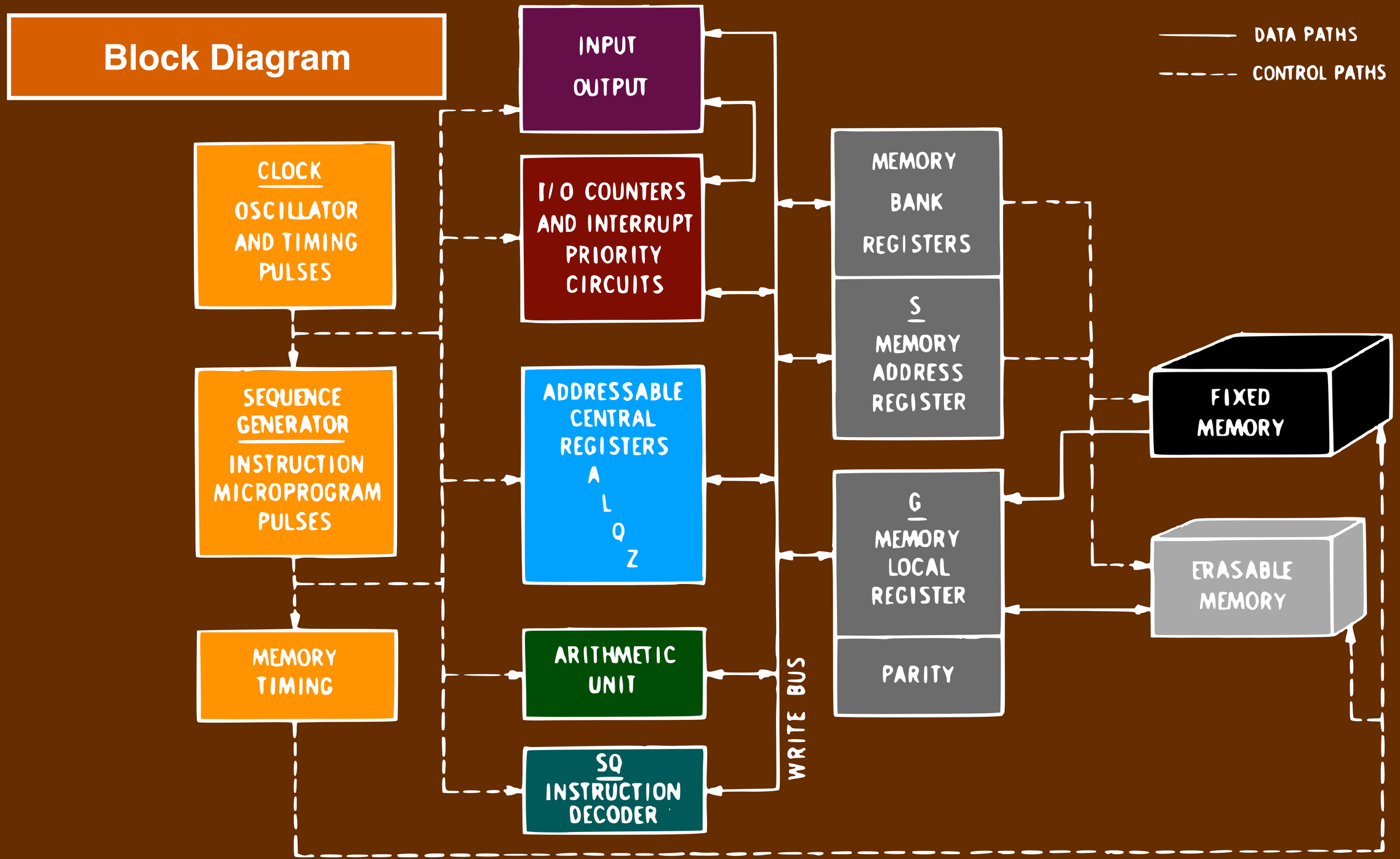
Block Diagram

Microcode

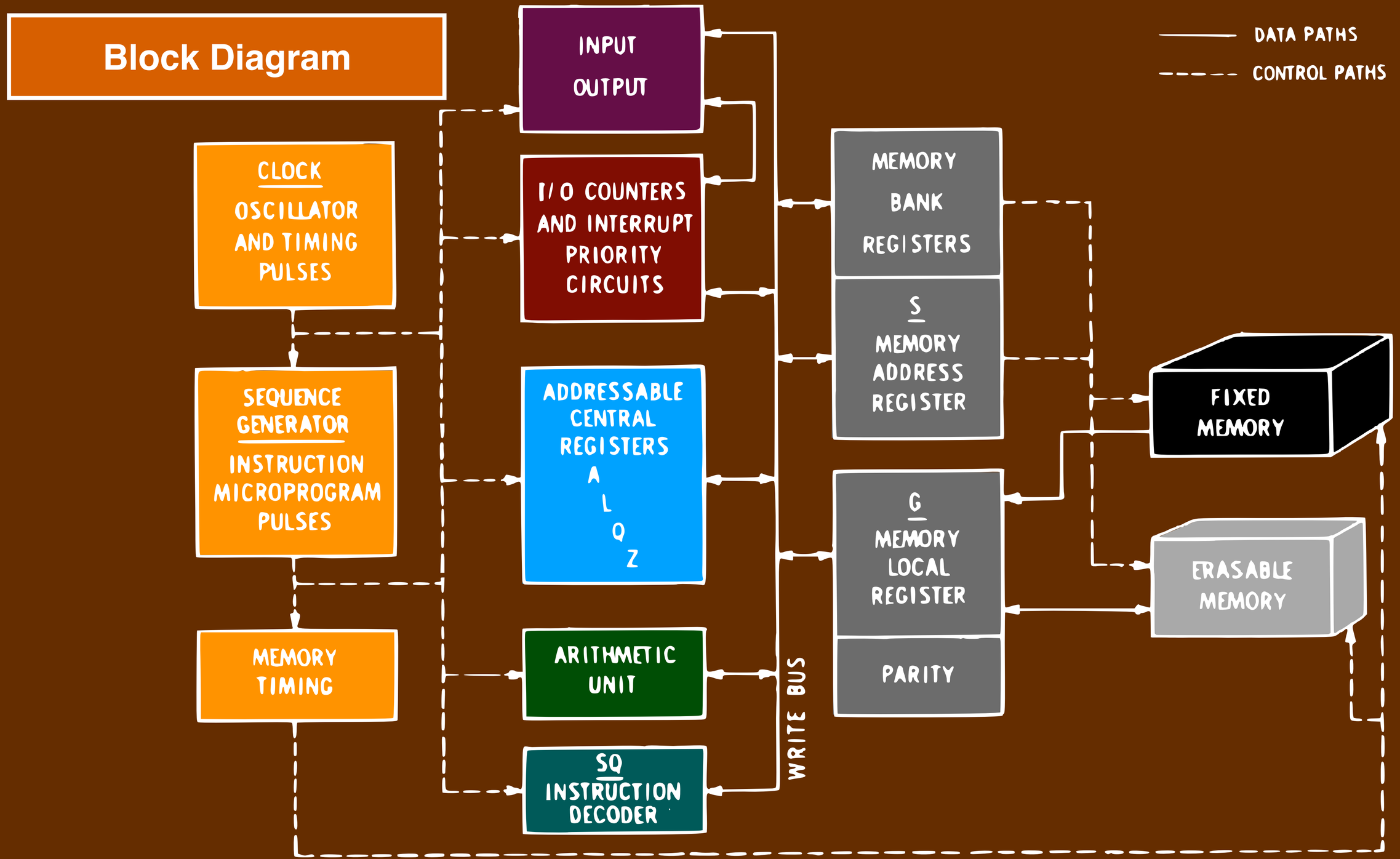
Hardware

- Block Diagram
- Schematics
- Microcode
- ICs
- Memory

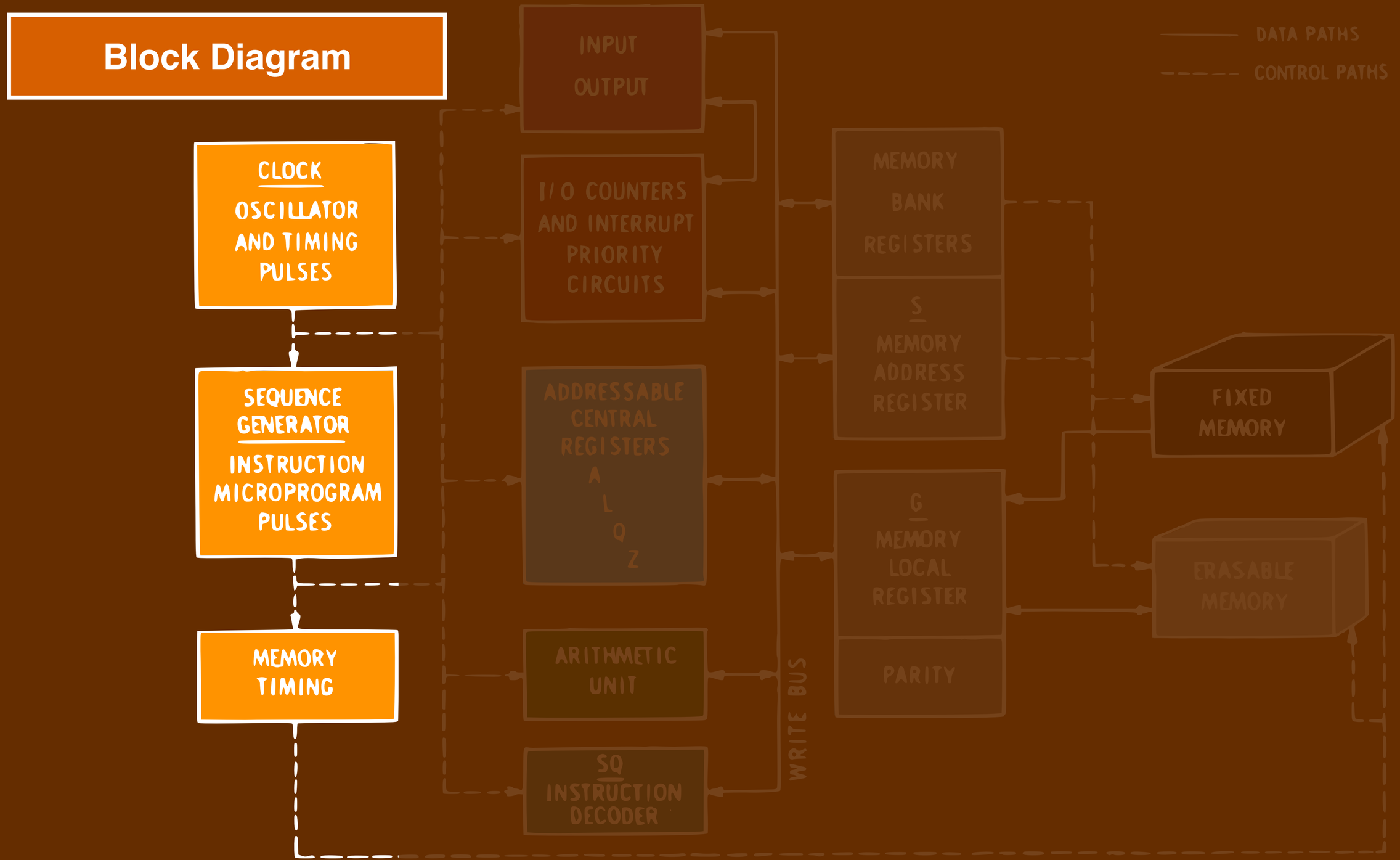
Block Diagram



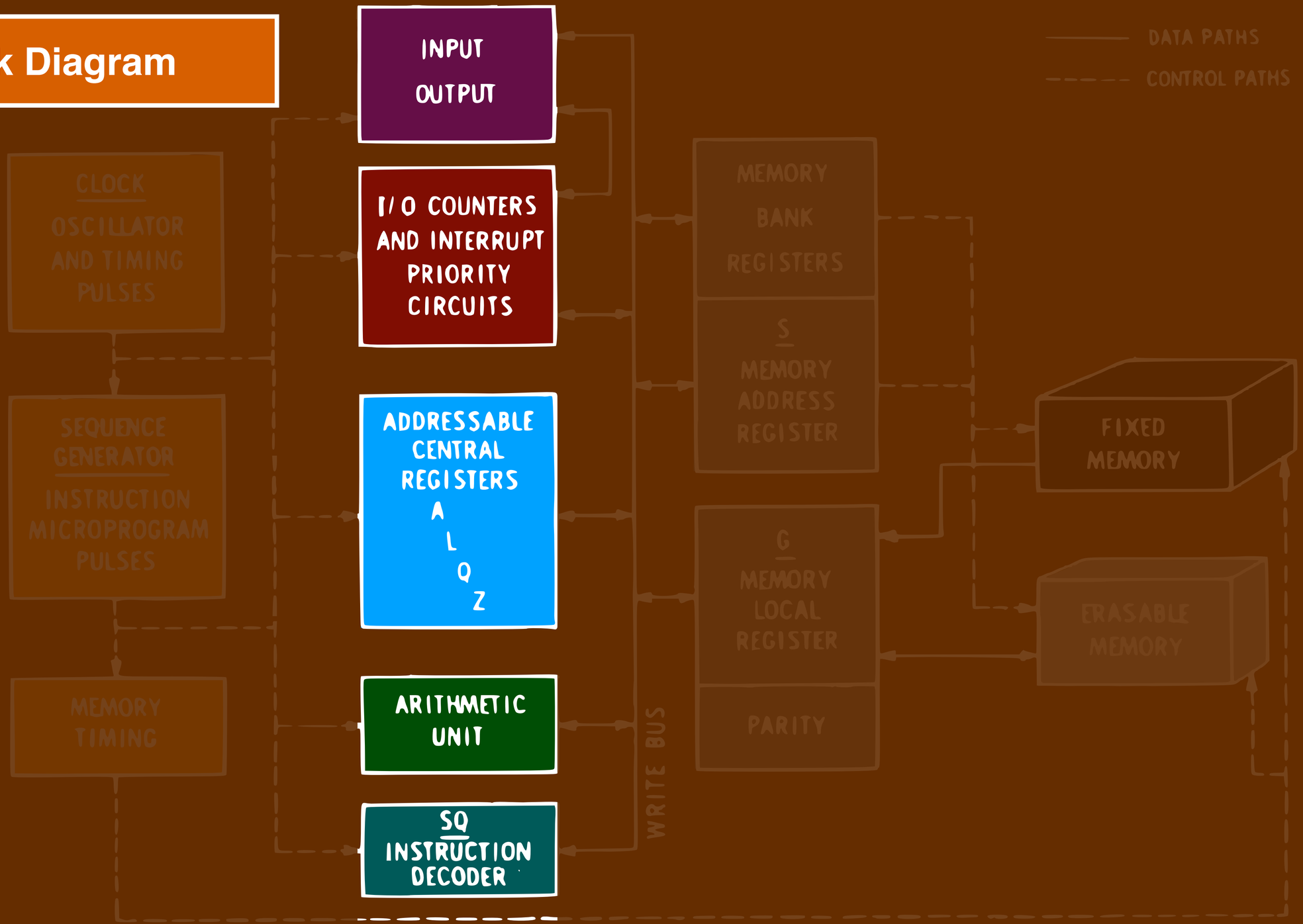
Block Diagram



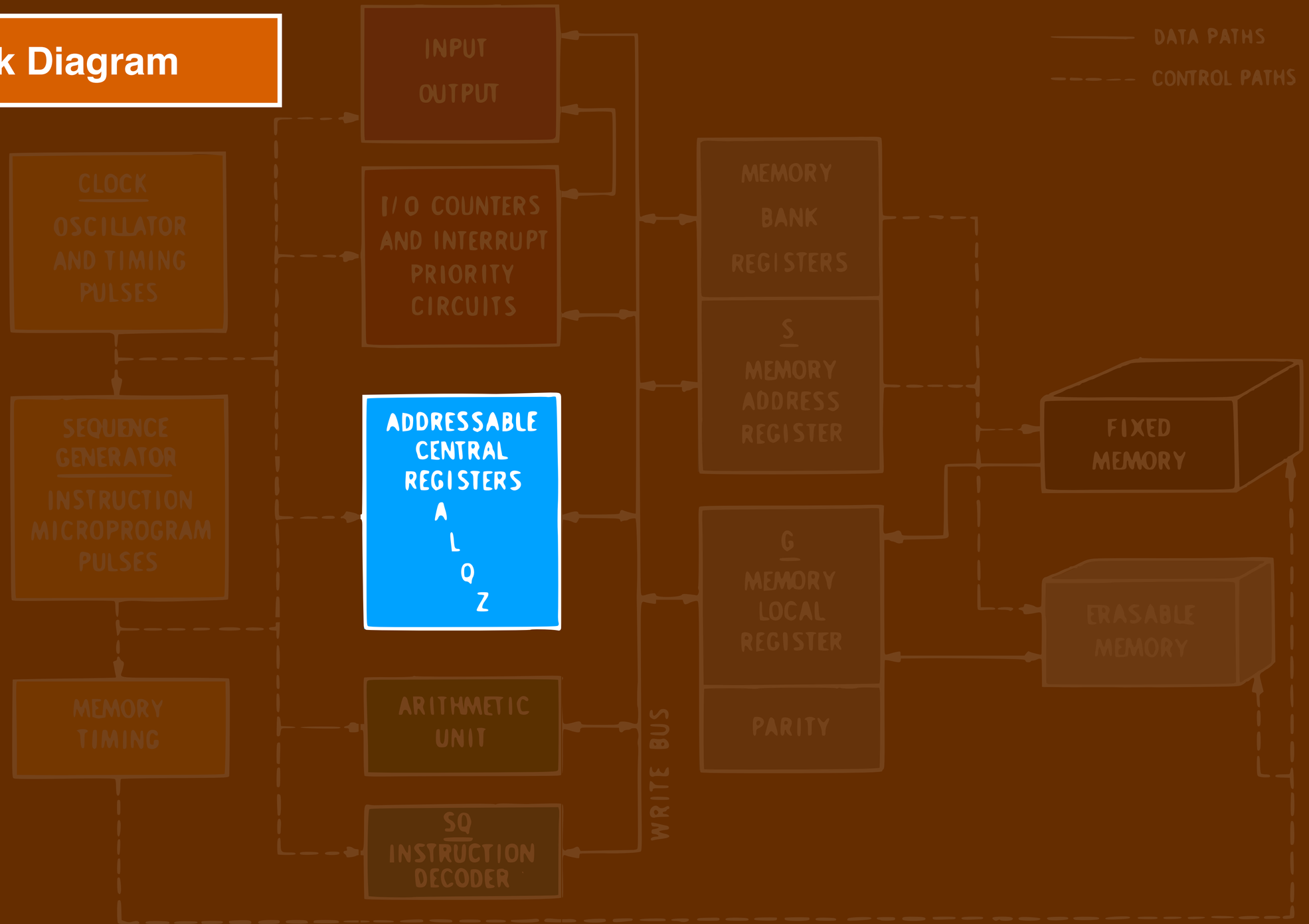
Block Diagram



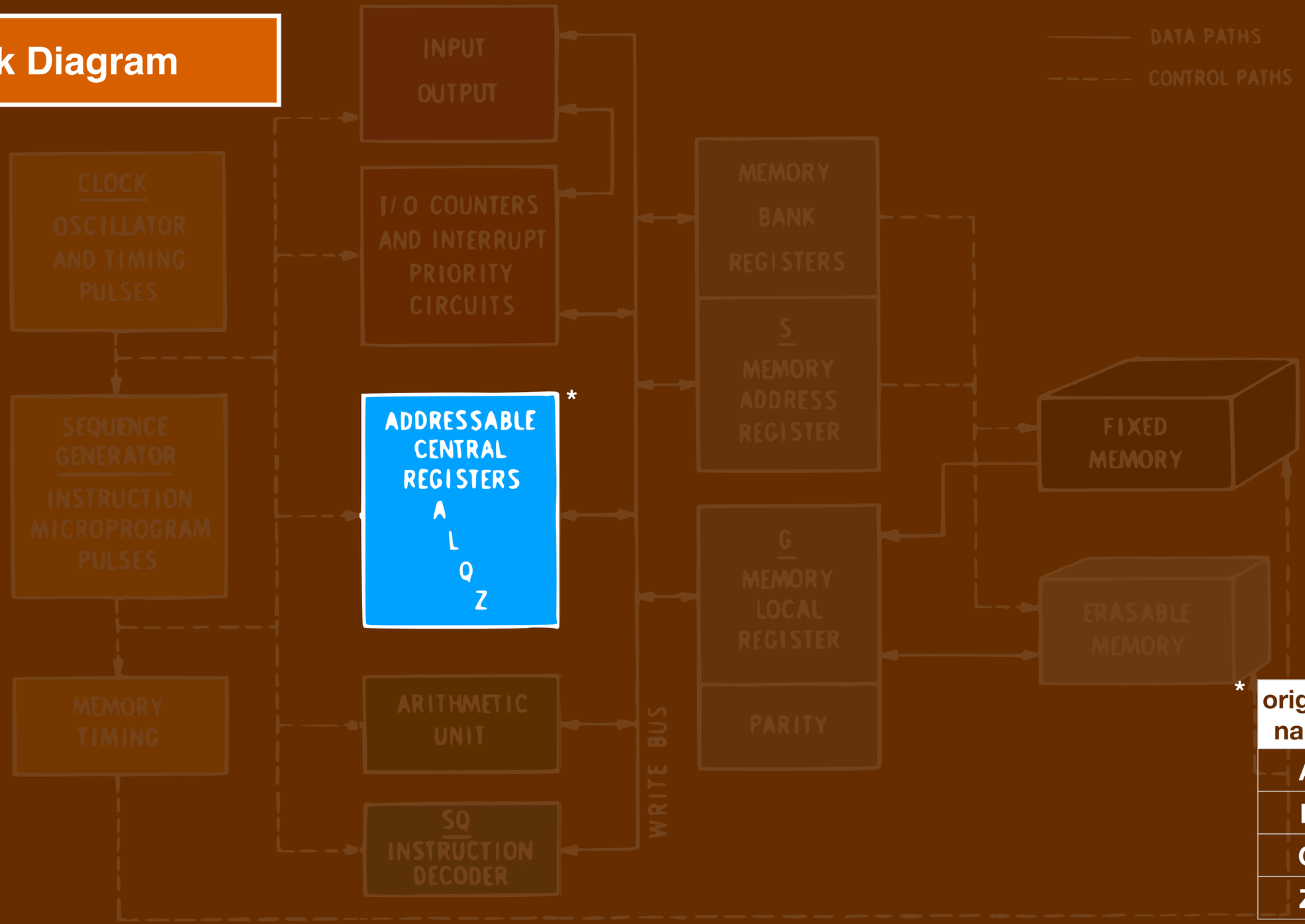
Block Diagram



Block Diagram



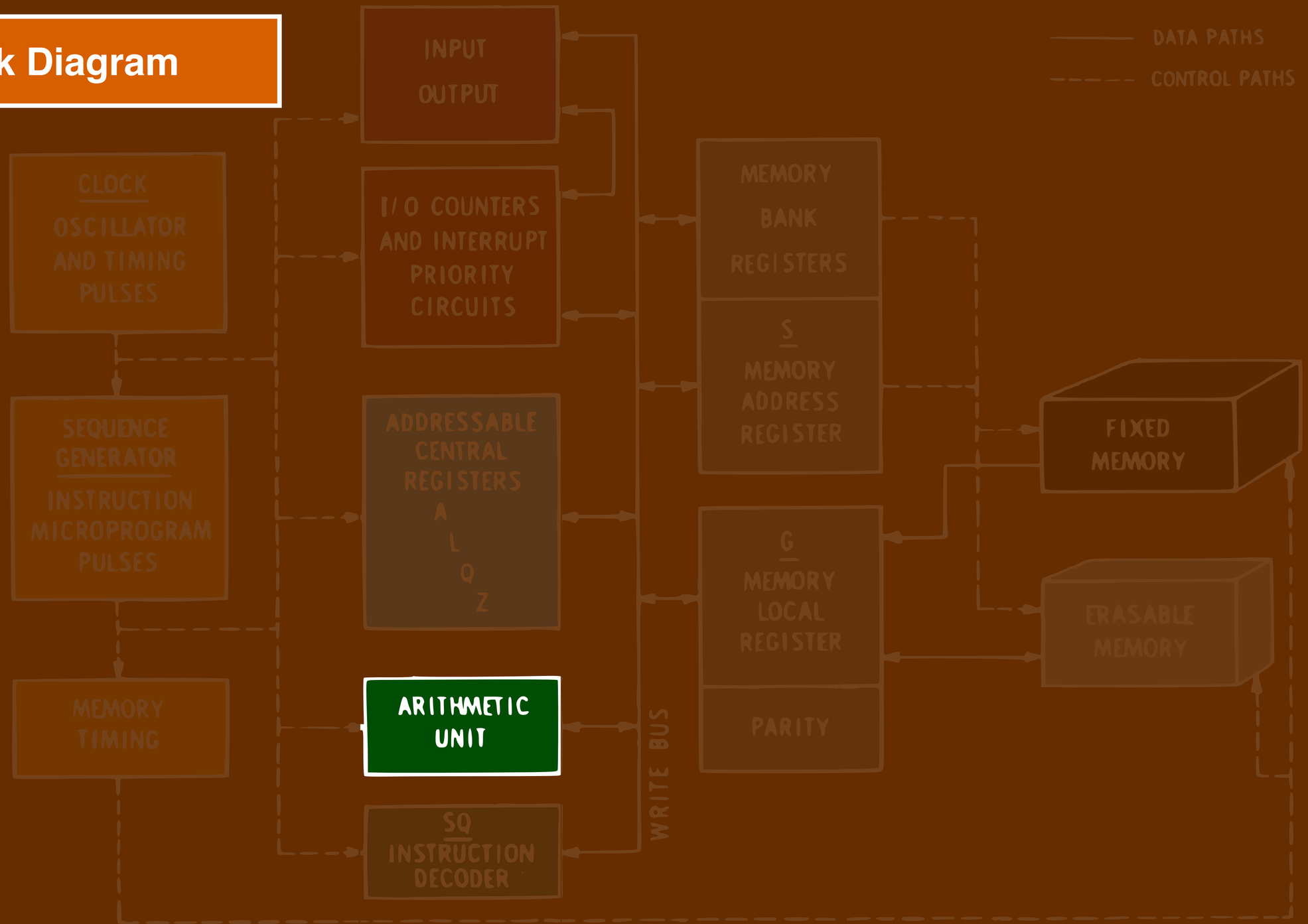
Block Diagram



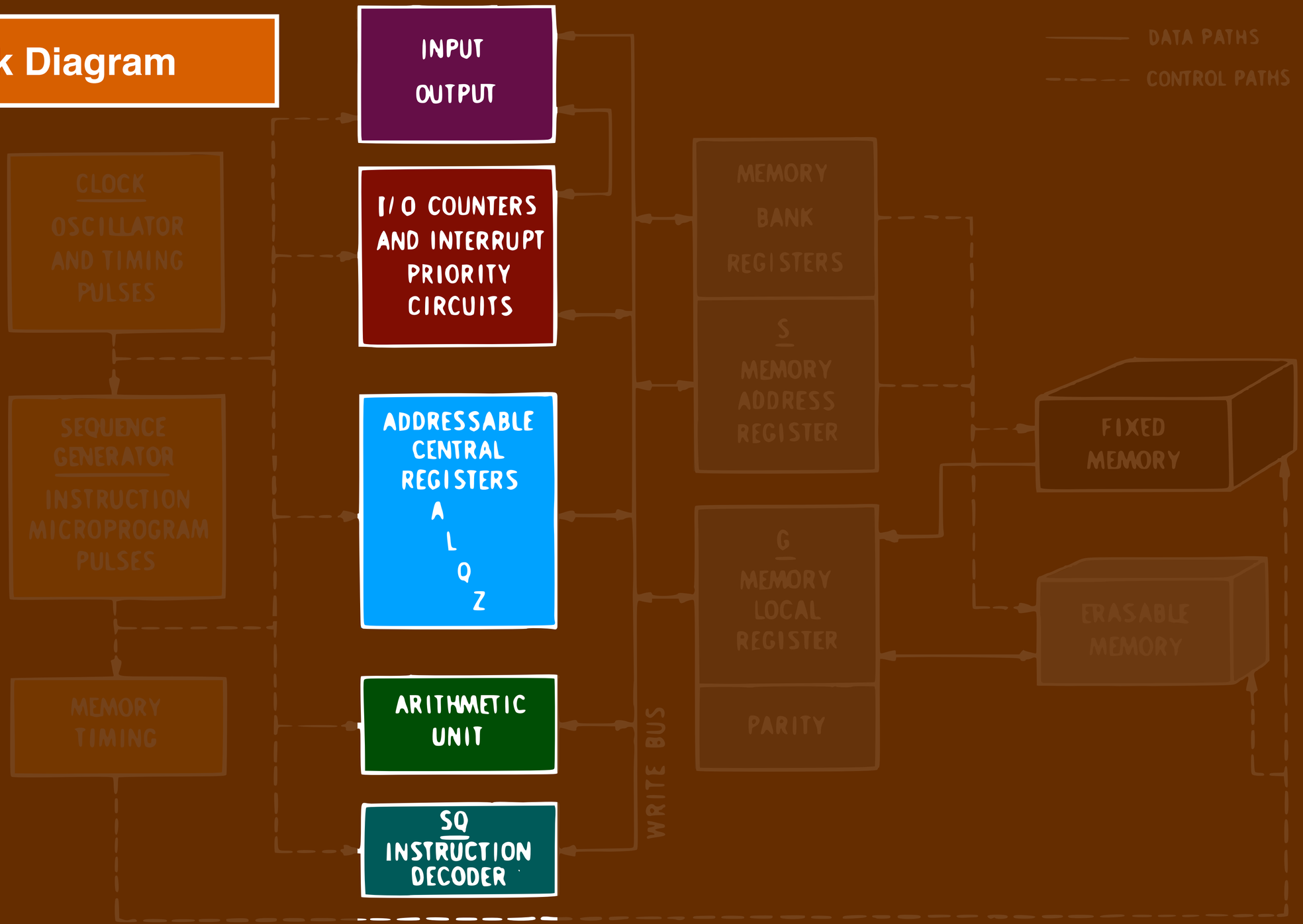
*

original name	my name
A	A
L	B
Q	LR
Z	PC

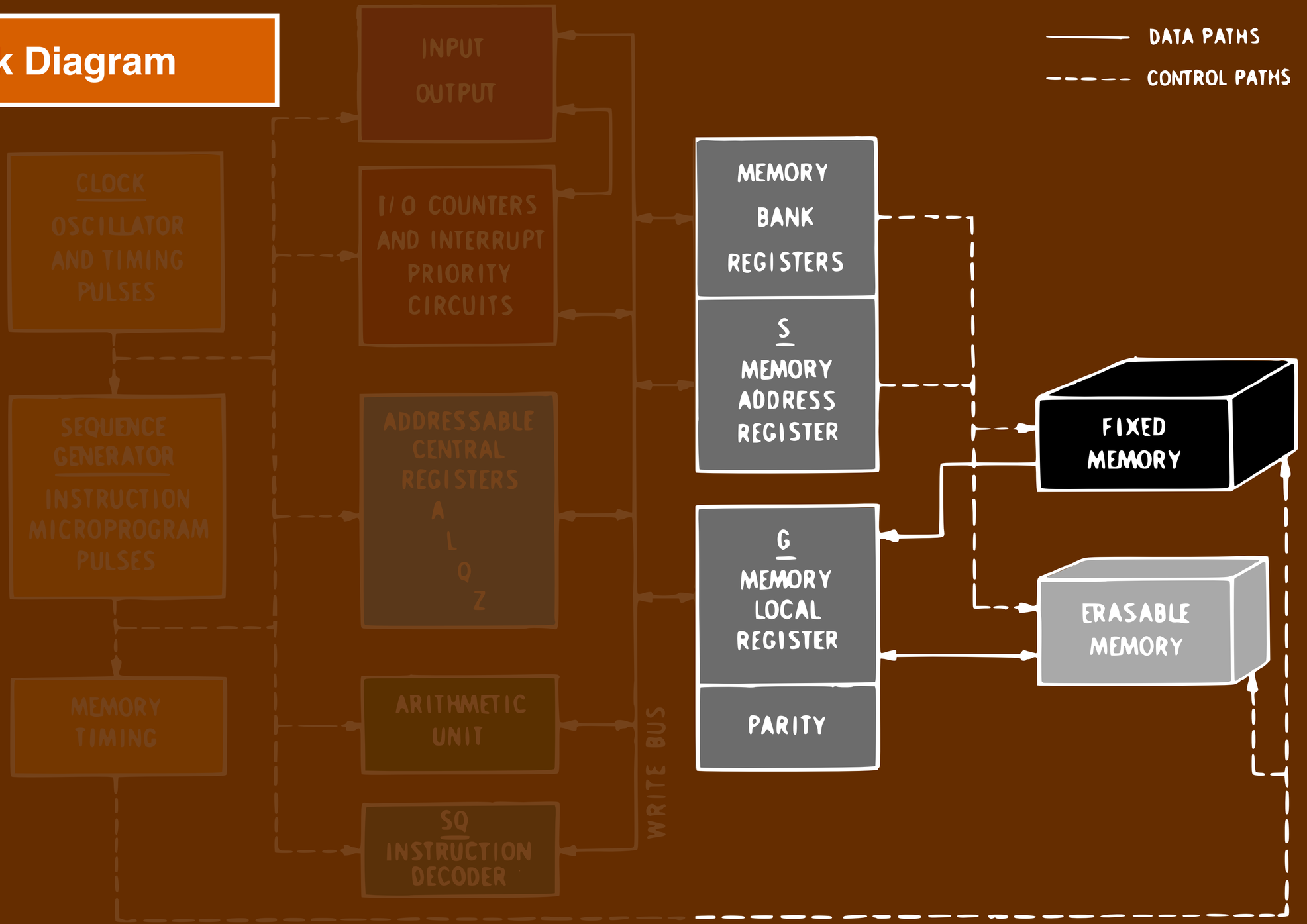
Block Diagram



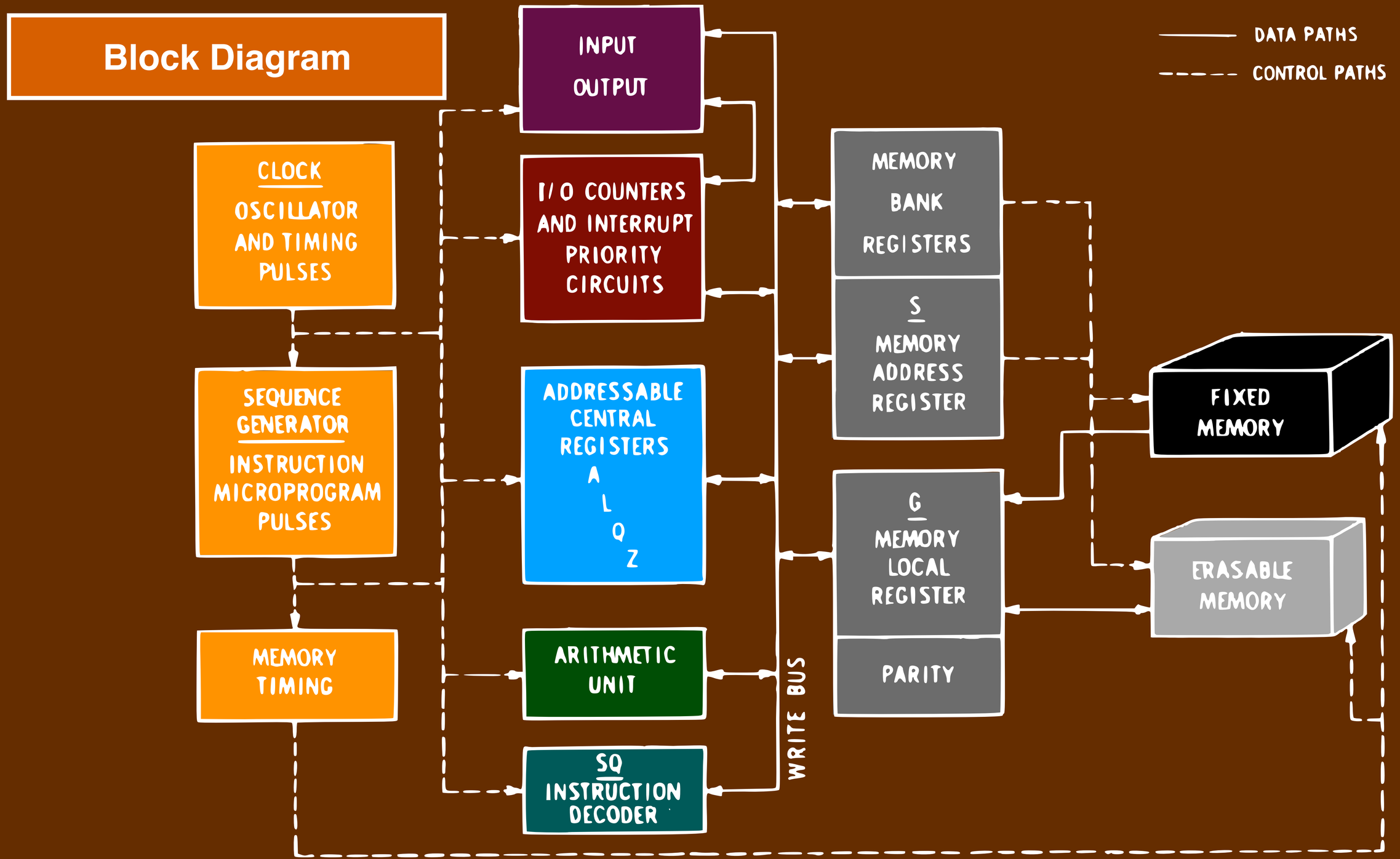
Block Diagram



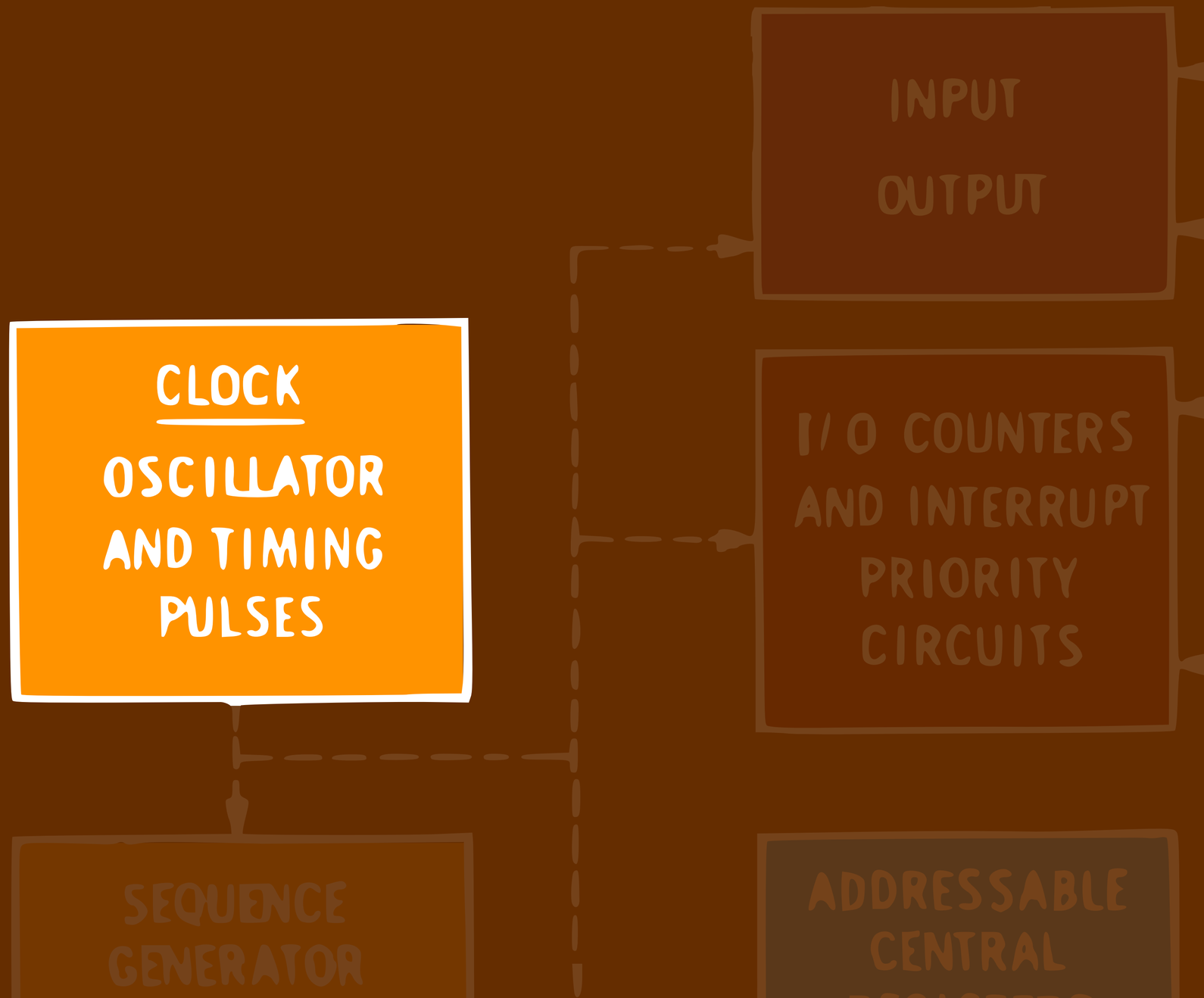
Block Diagram



Block Diagram

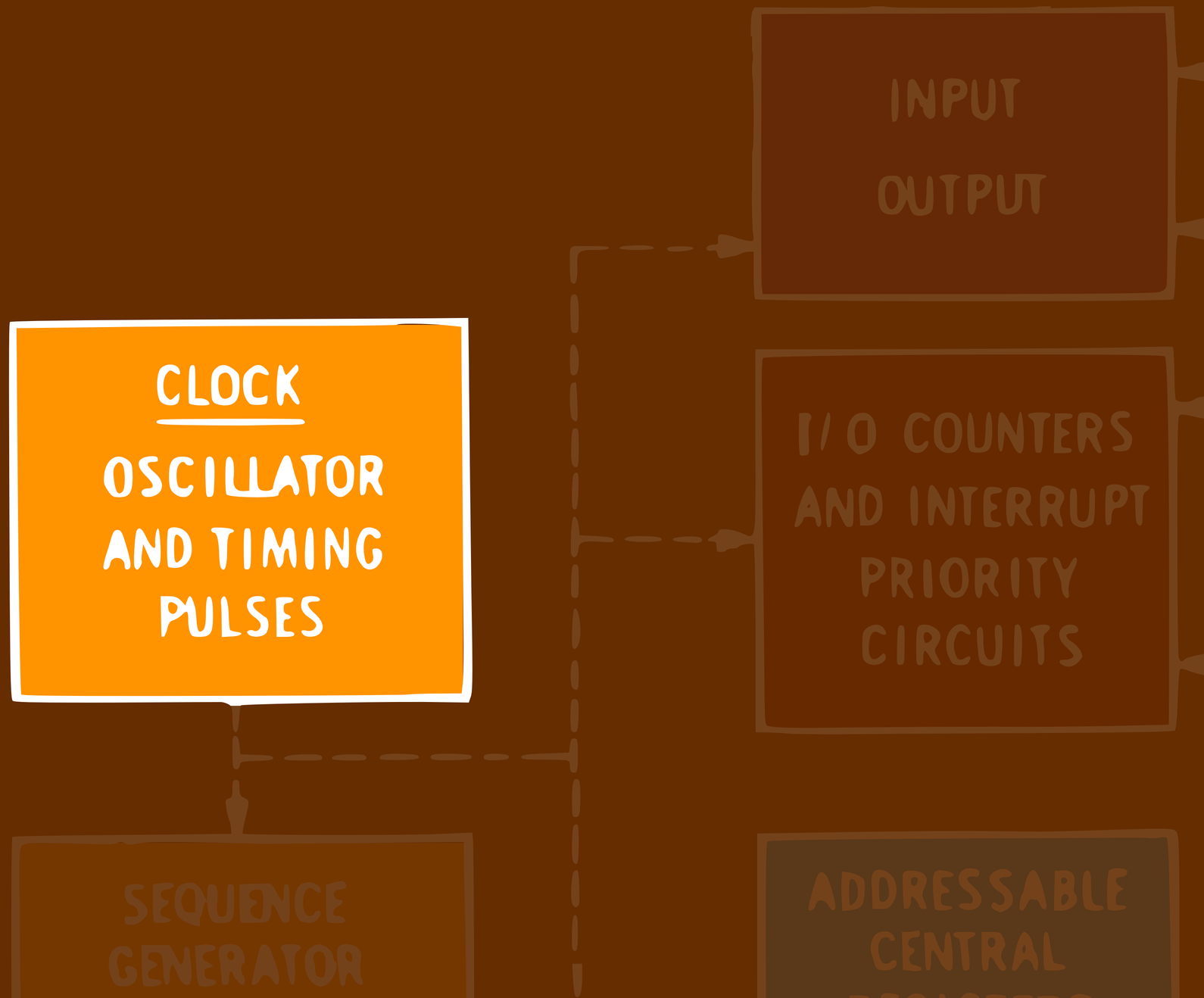


Clock

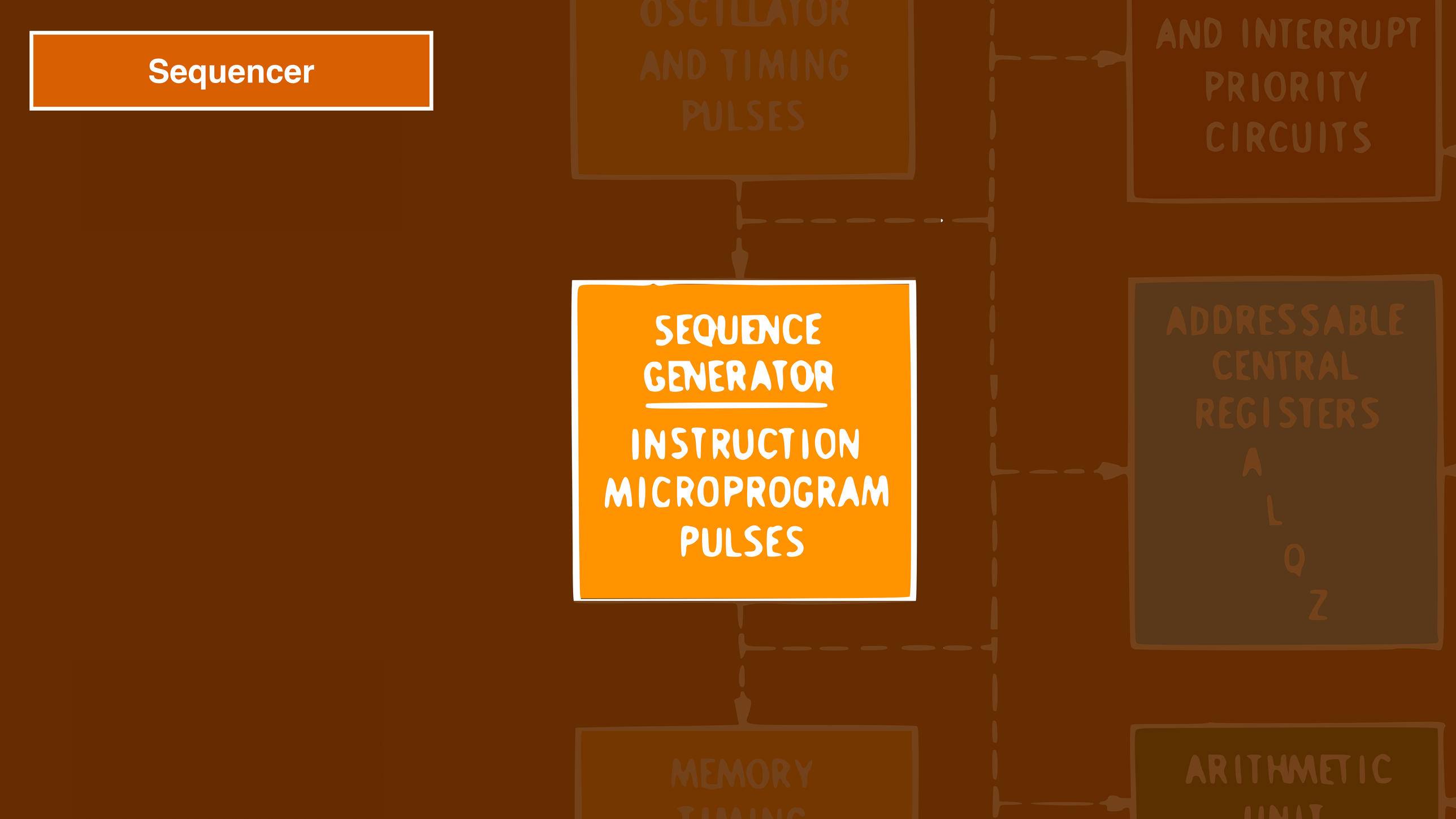

CLOCK
OSCILLATOR AND TIMING PULSES
INPUT
OUTPUT
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
SEQUENCE GENERATOR
ADDRESSABLE CENTRAL

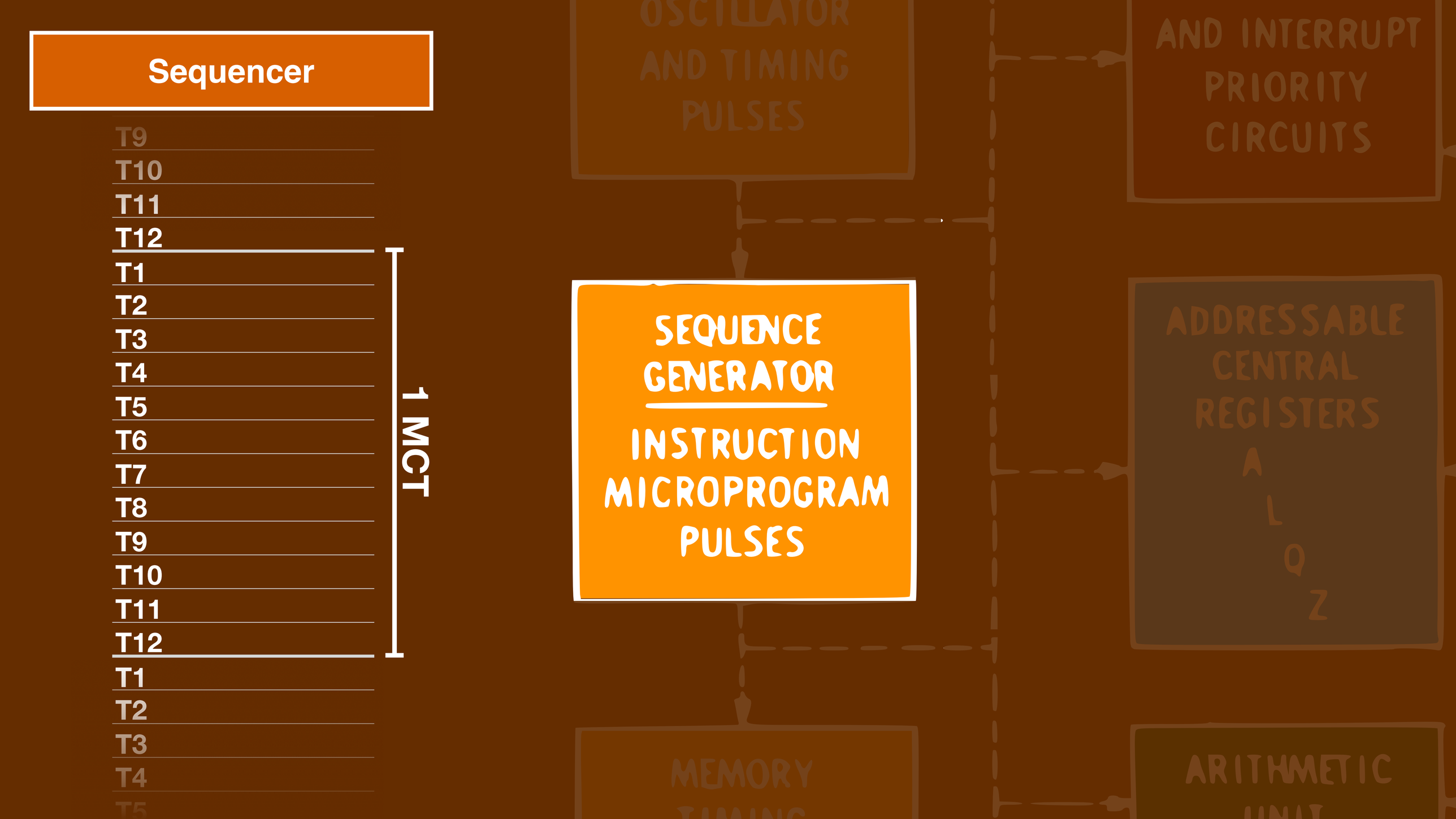
Clock

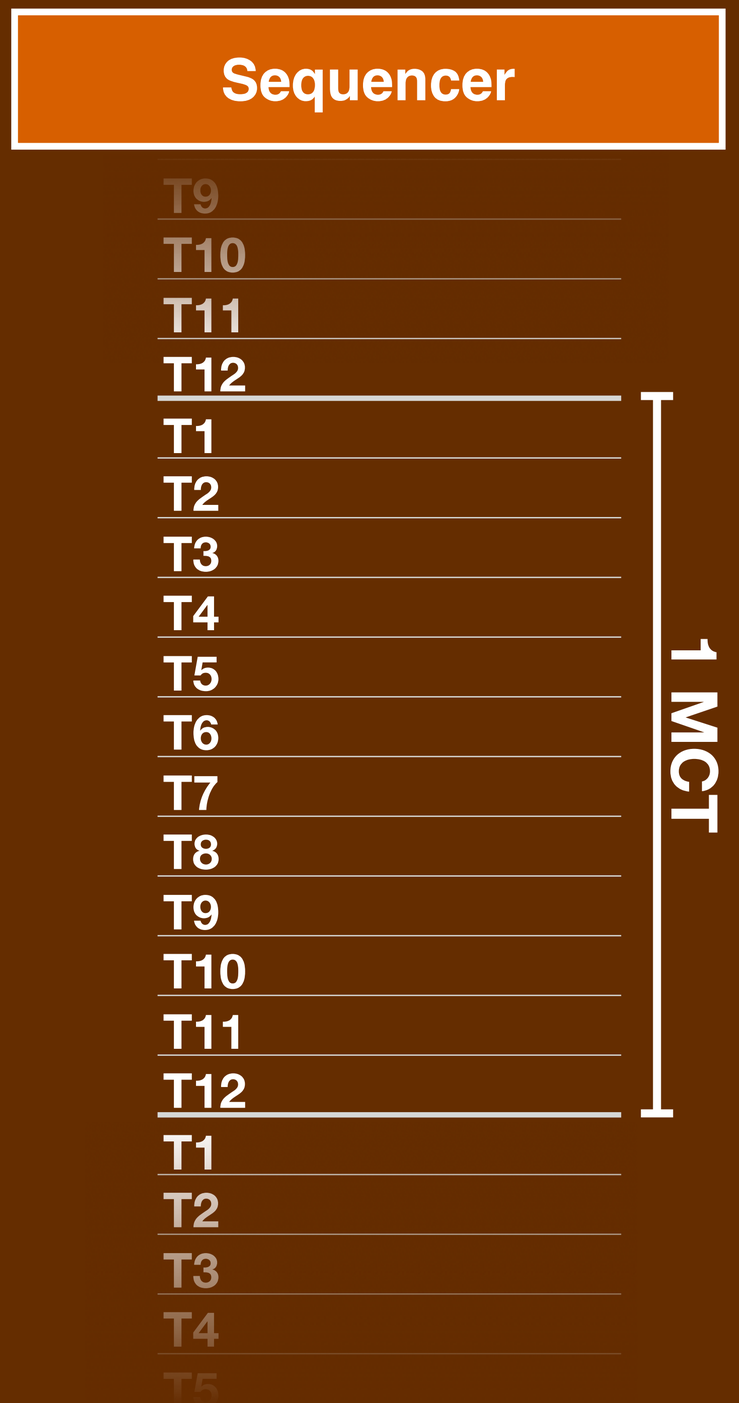
1.024 MHz

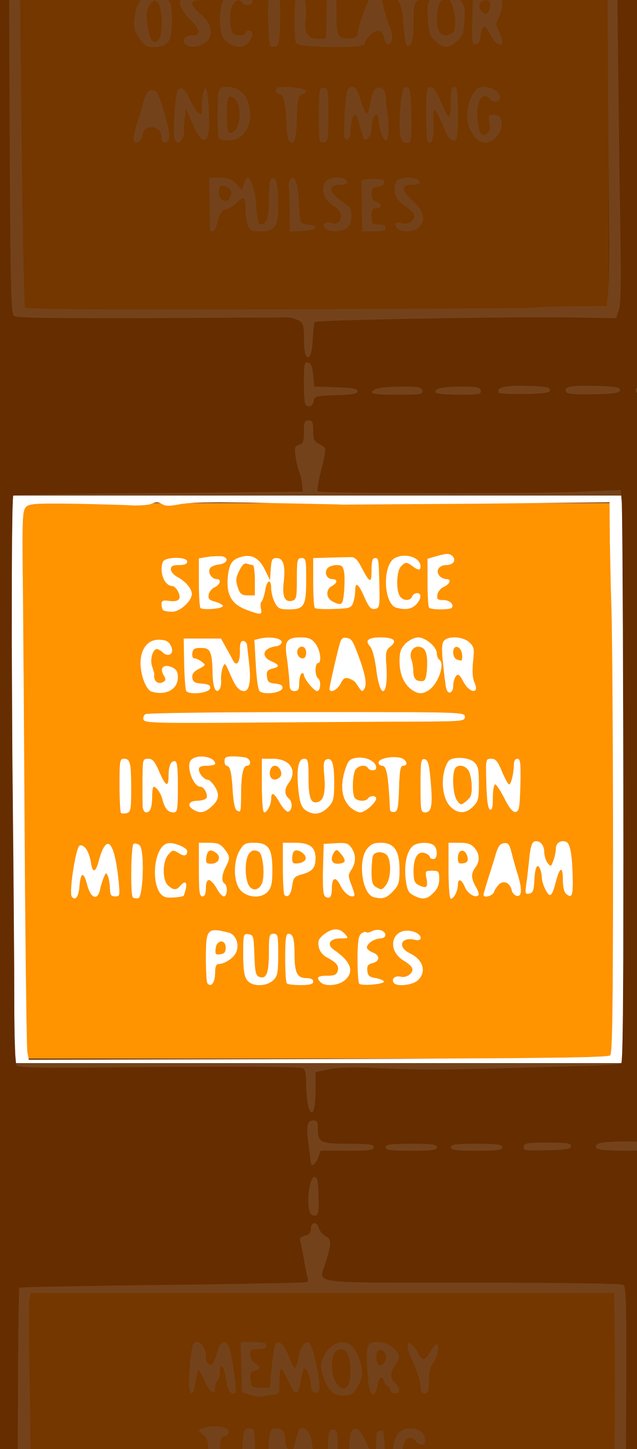


Sequencer

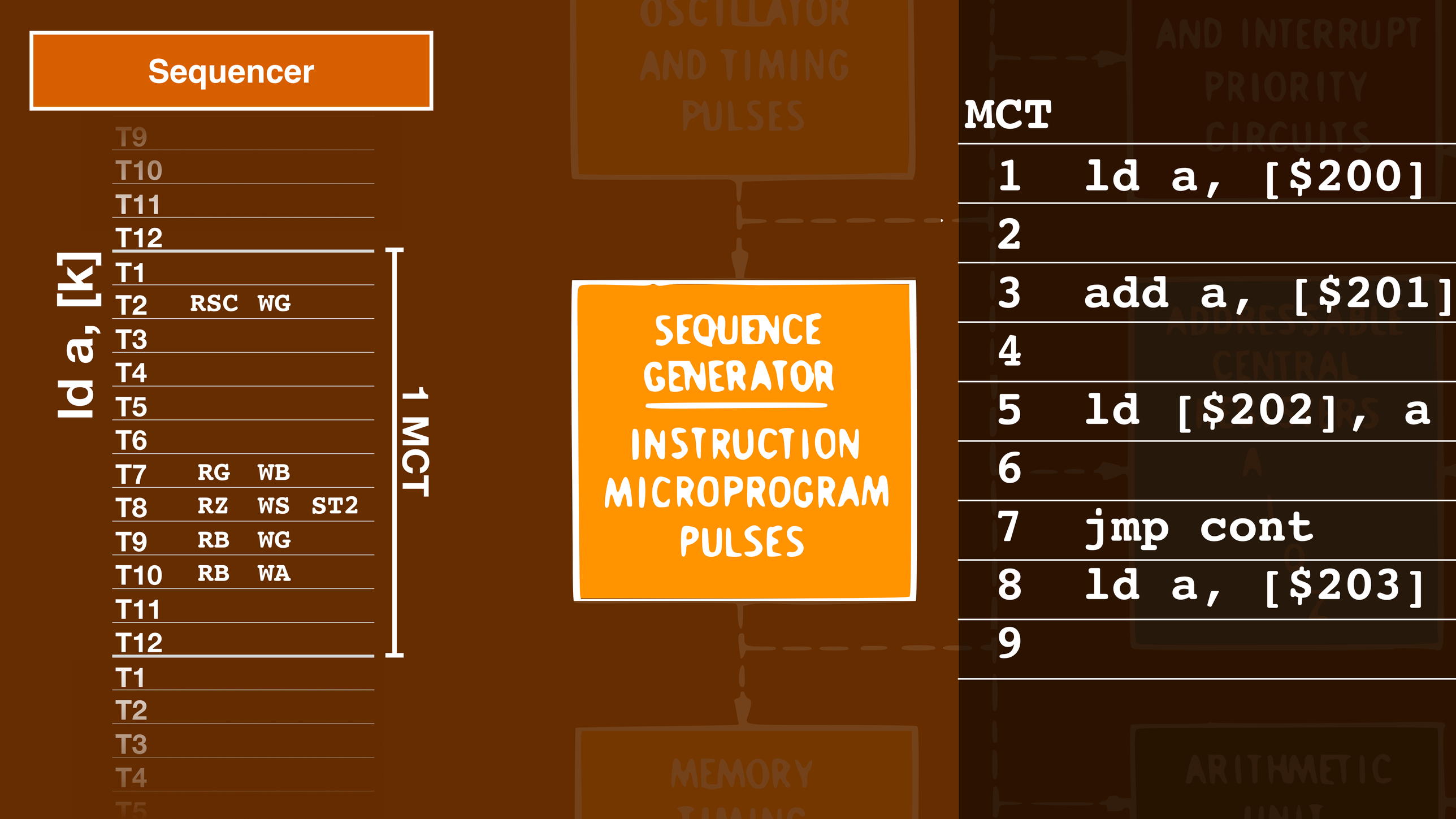

OSCILLATOR AND TIMING PULSES
AND INTERRUPT PRIORITY CIRCUITS
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
MEMORY
ARITHMETIC

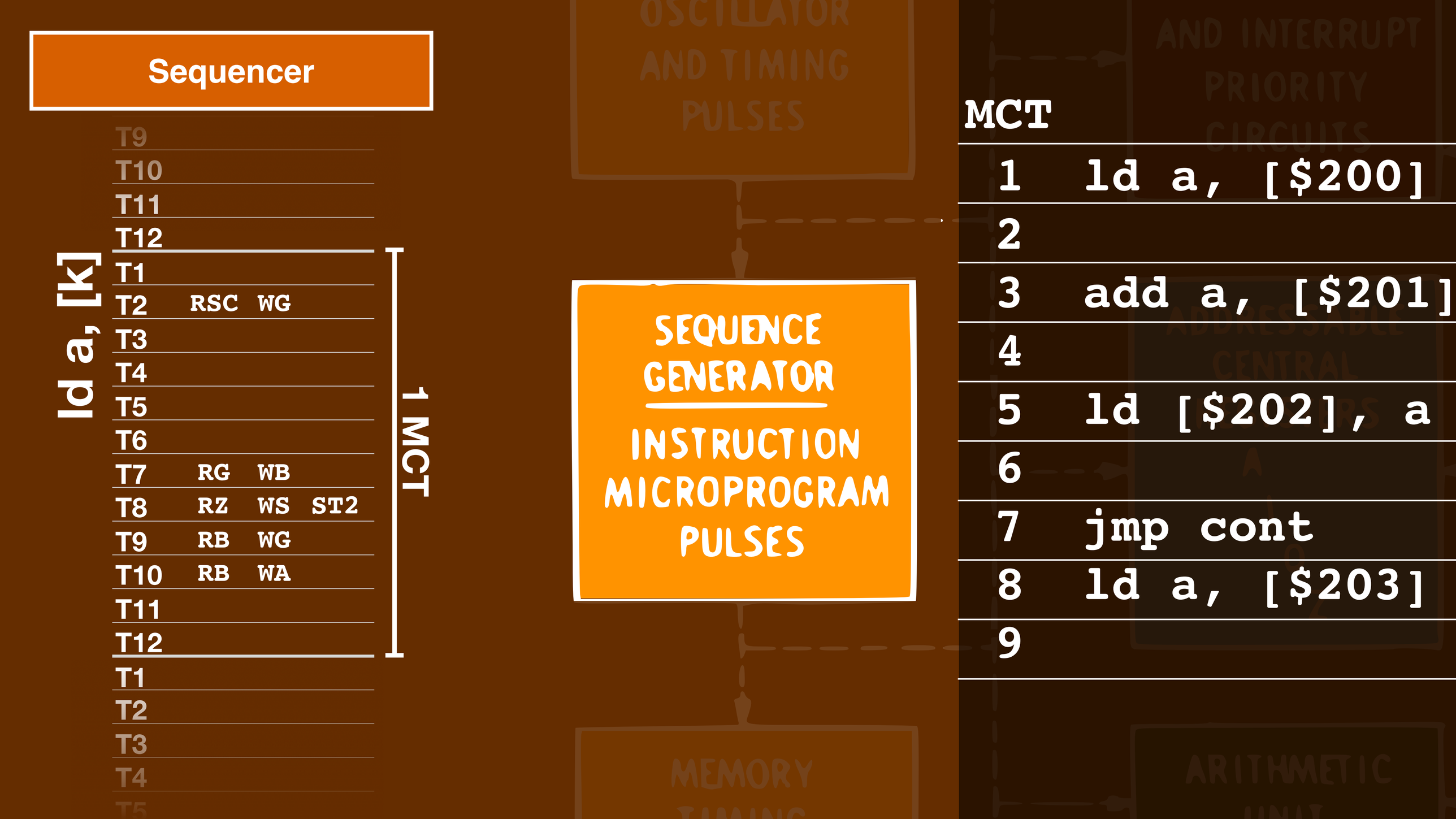

Sequencer
T9
T10
T11
T12
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
T1
T2
T3
T4
1 MCT
OSCILLATOR AND TIMING PULSES
AND INTERRUPT PRIORITY CIRCUITS
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
MEMORY
ARITHMETIC

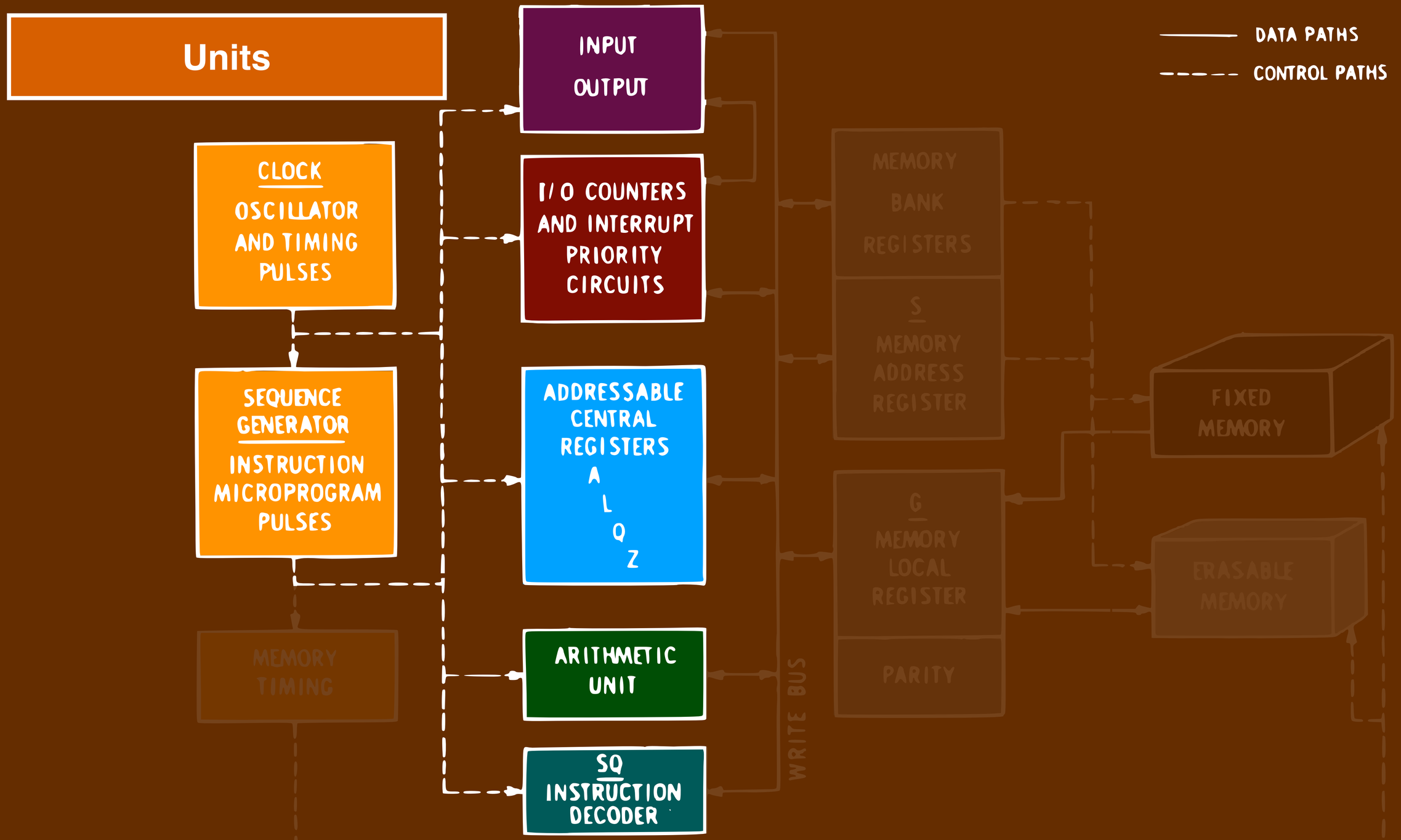


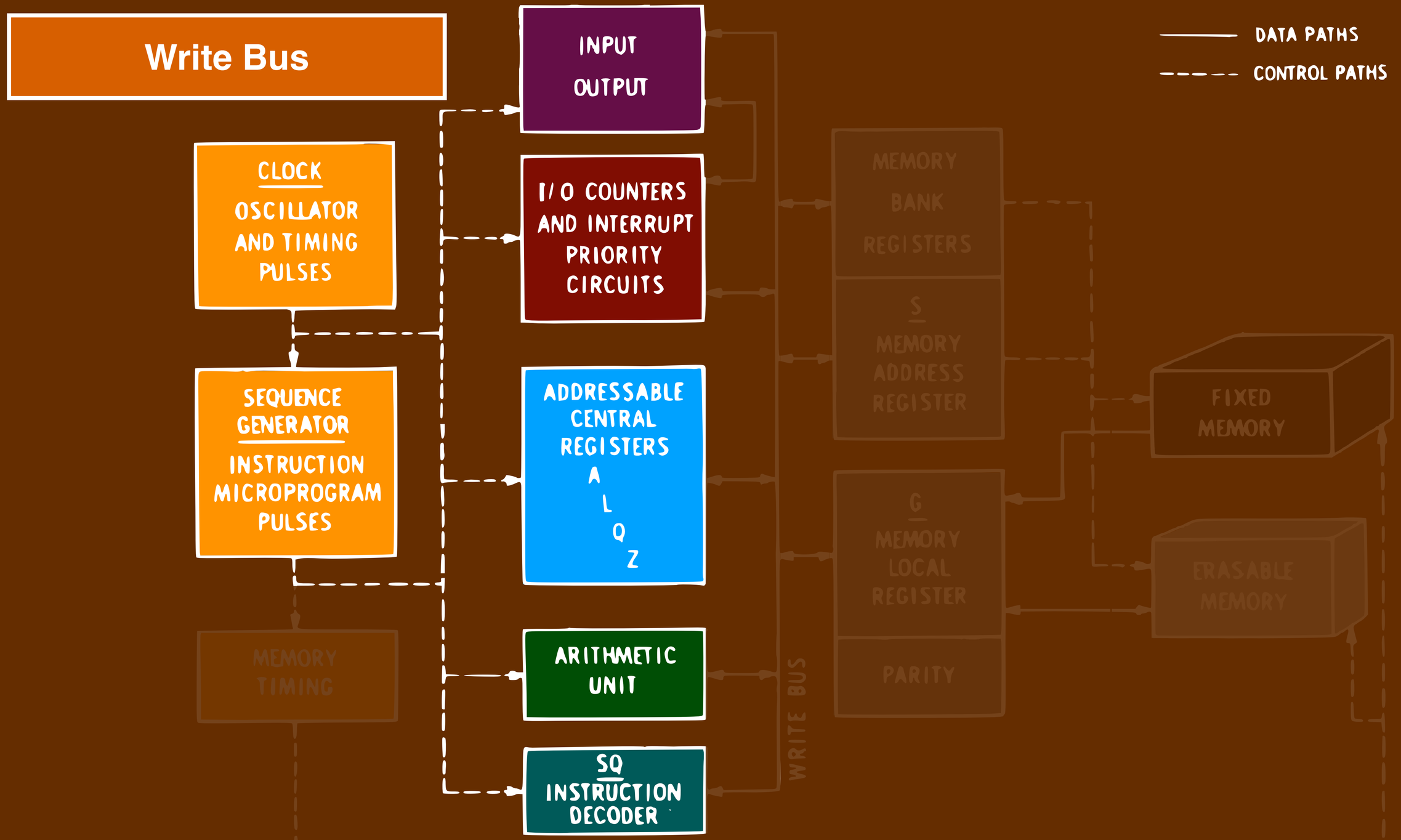


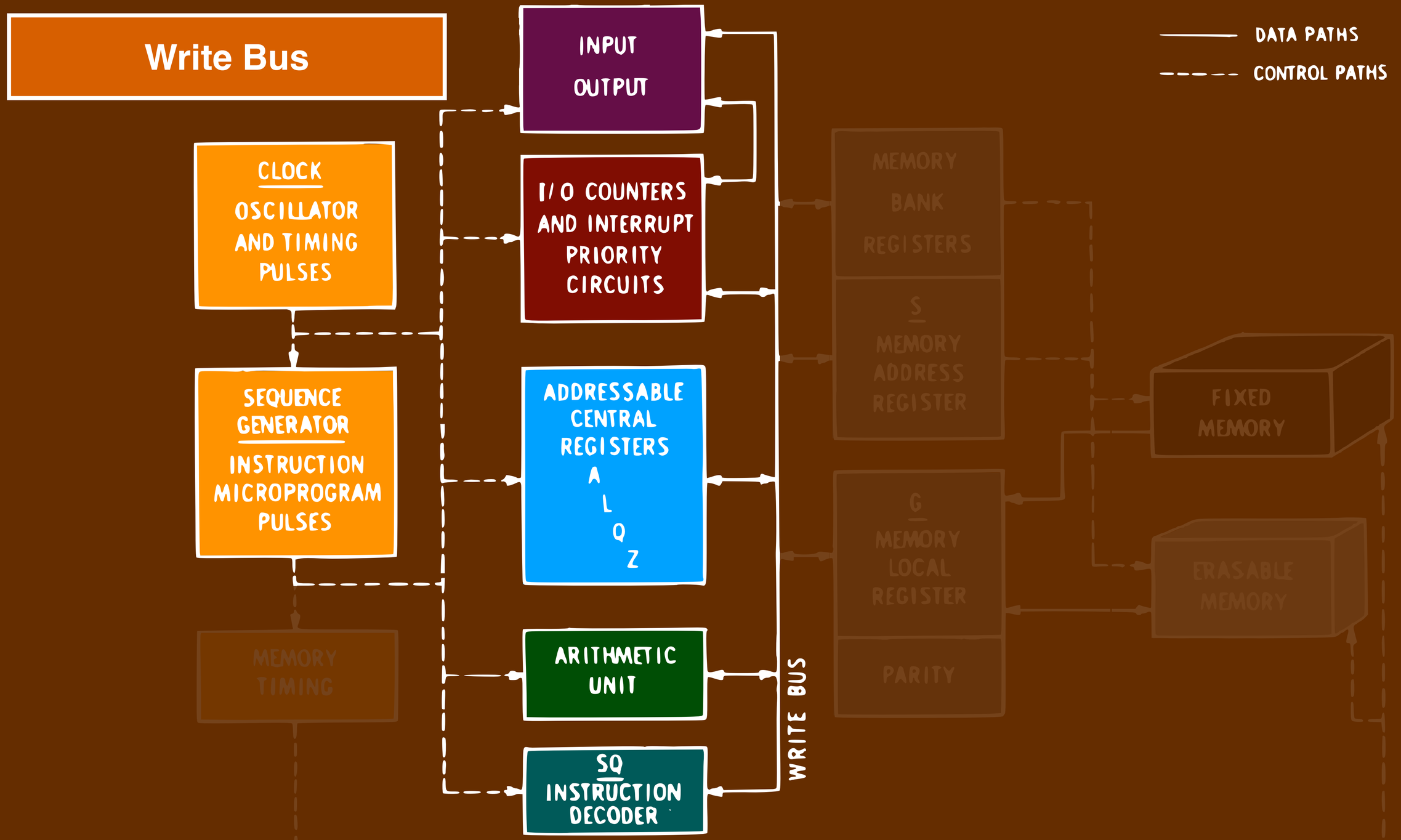
MCT	
1	ld a, [$200]
2	
3	add a, [$201]
4	
5	ld [$202], a
6	
7	jmp cont
8	ld a, [$203]
9	

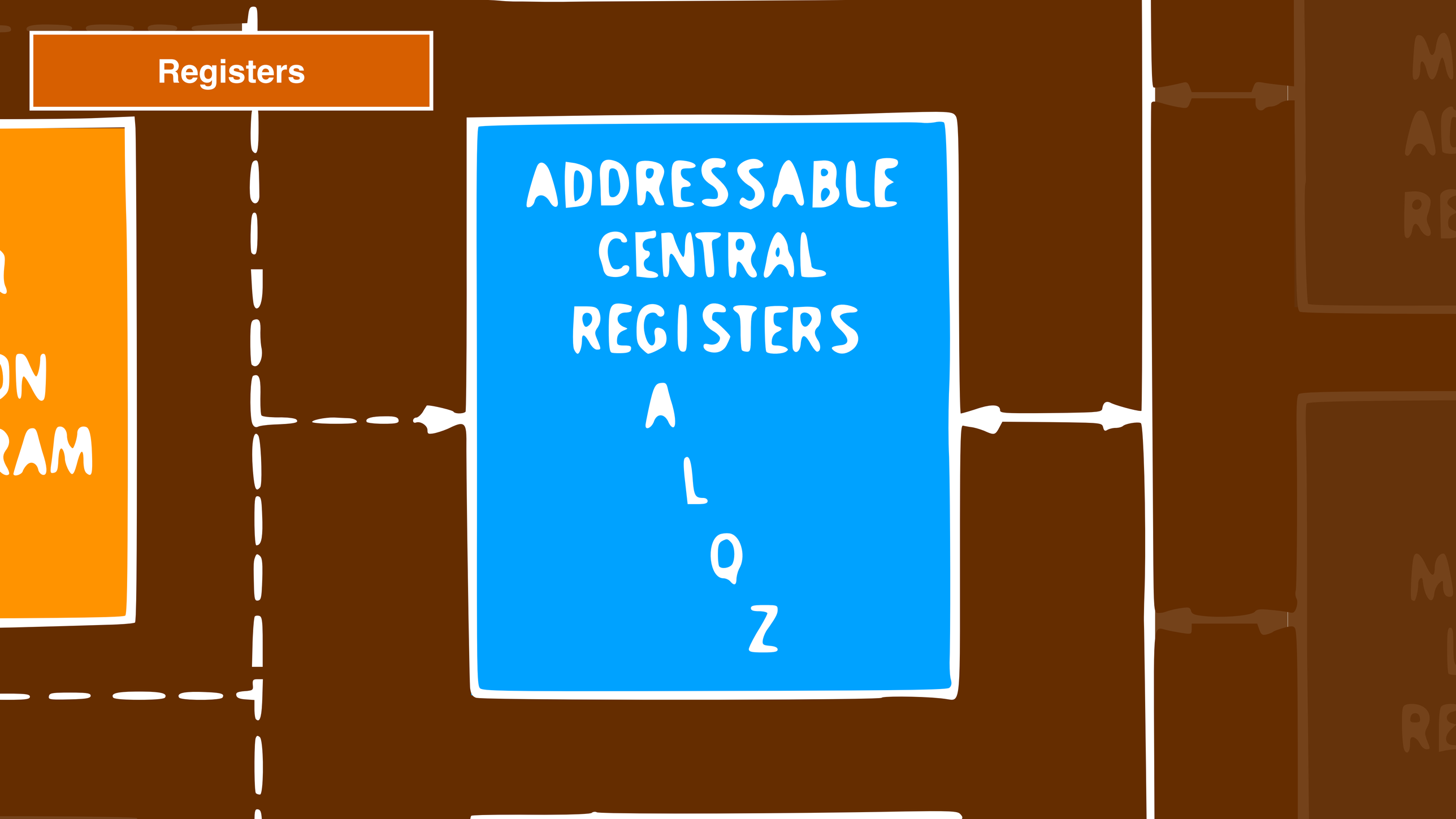

Sequencer
T9
T10
T11
T12
ld a, [k]
T1
T2 RSC WG
T3
T4
T5
T6
T7 RG WB
T8 RZ WS ST2
T9 RB WG
T10 RB WA
T11
T12
1 MCT
T1
T2
T3
T4
T5
OSCILLATOR AND TIMING PULSES
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
MEMORY
AND INTERRUPT PRIORITY CIRCUITS
ADDRESS BUS CENTRAL
ARITHMETIC
MCT
1 ld a, [$200]
2
3 add a, [$201]
4
5 ld [$202], a
6
7 jmp cont
8 ld a, [$203]
9

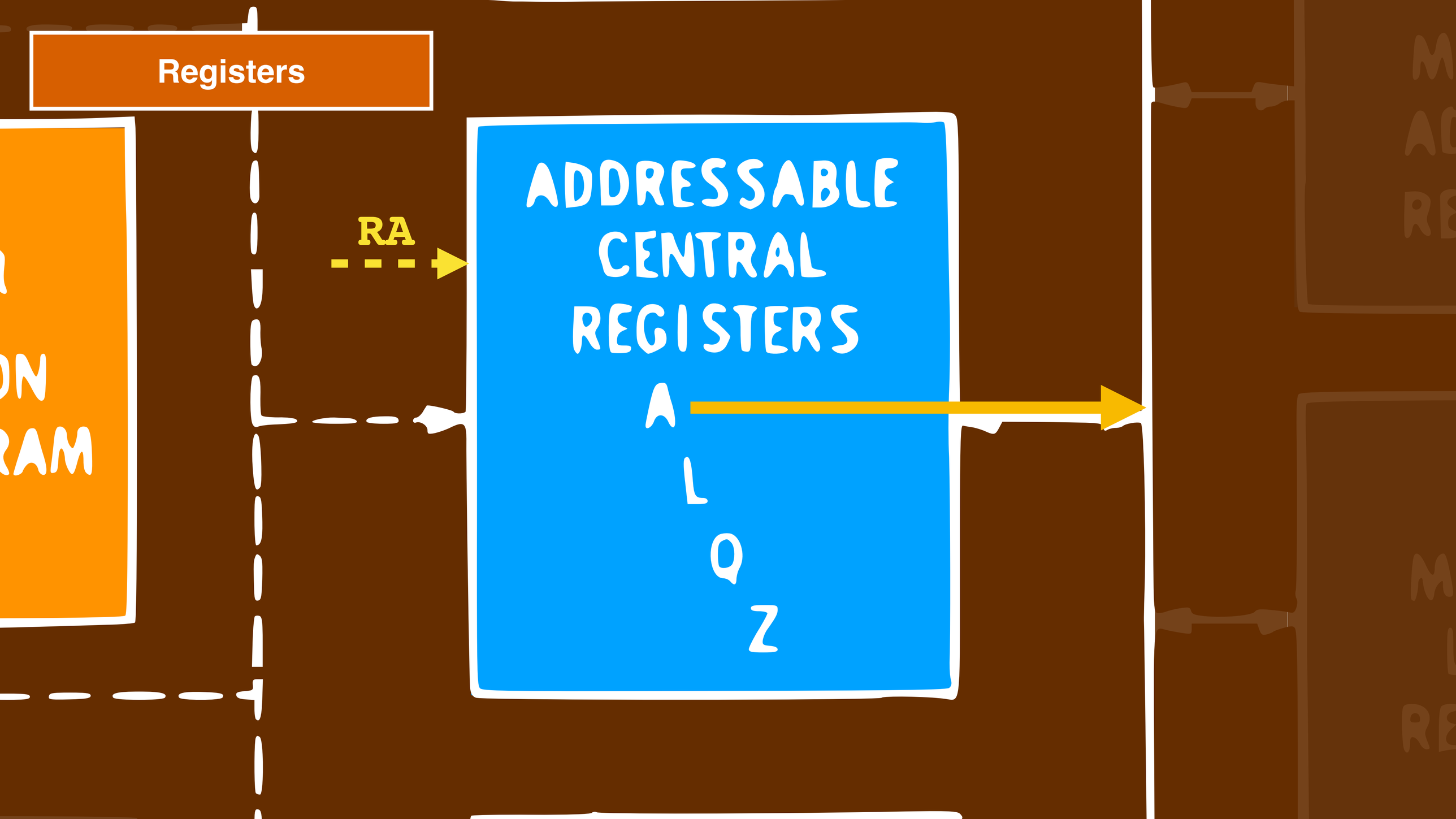

Sequencer
T9
T10
T11
T12
ld a, [k]
T1
T2 RSC WG
T3
T4
T5
T6
T7 RG WB
T8 RZ WS ST2
T9 RB WG
T10 RB WA
T11
T12
1 MCT
T1
T2
T3
T4
T5
OSCILLATOR AND TIMING PULSES
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
MEMORY
AND INTERRUPT PRIORITY CIRCUITS
ADDRESS BUS CENTRAL
ARITHMETIC
MCT
1 ld a, [$200]
2
3 add a, [$201]
4
5 ld [$202], a
6
7 jmp cont
8 ld a, [$203]
9

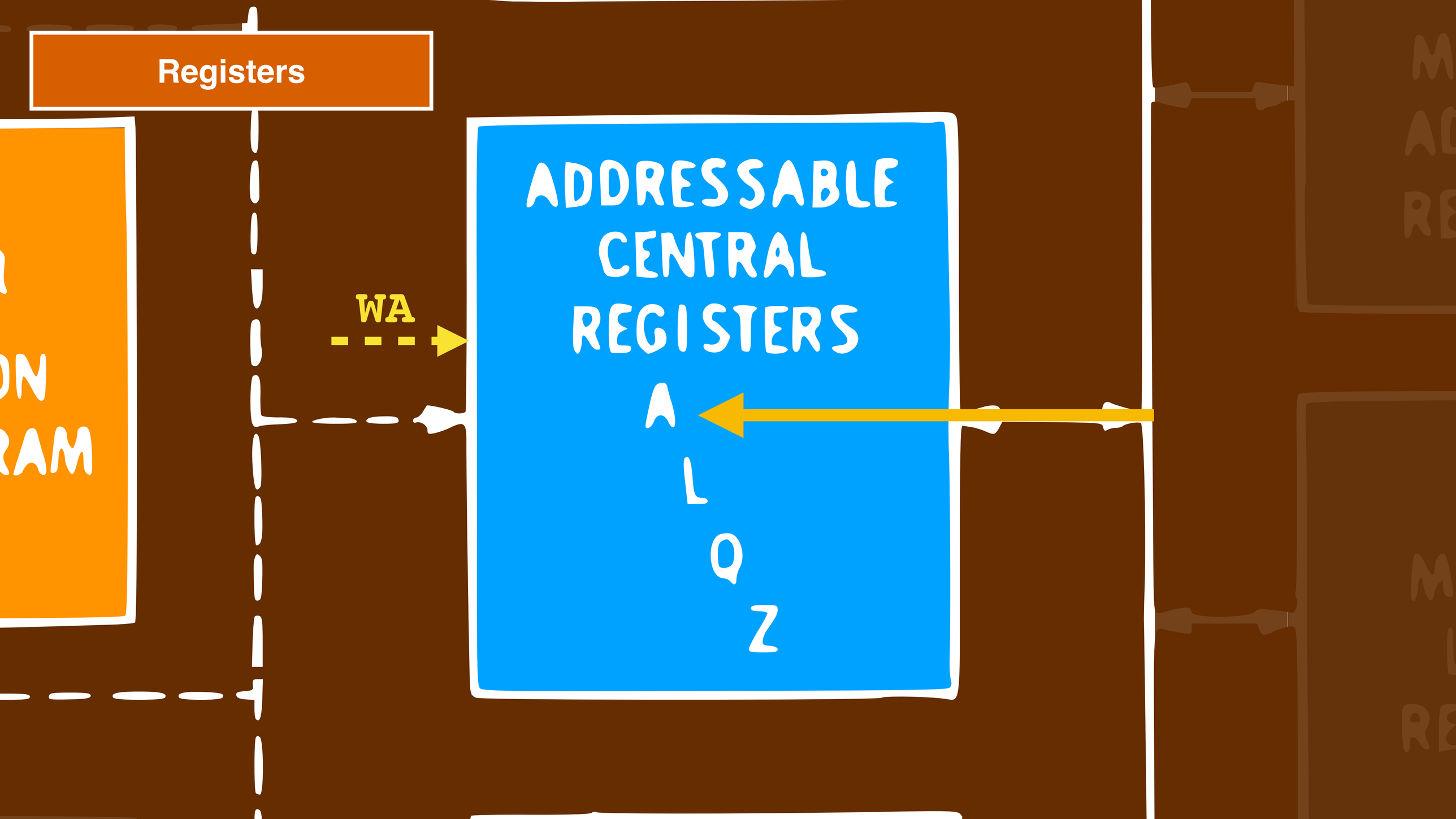

Units
INPUT
OUTPUT
CLOCK
OSCILLATOR AND TIMING PULSES
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
MEMORY TIMING
ARITHMETIC UNIT
SQ
INSTRUCTION DECODER
WRITE BUS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
G
MEMORY LOCAL REGISTER
PARITY
FIXED MEMORY
ERASABLE MEMORY
DATA PATHS
CONTROL PATHS

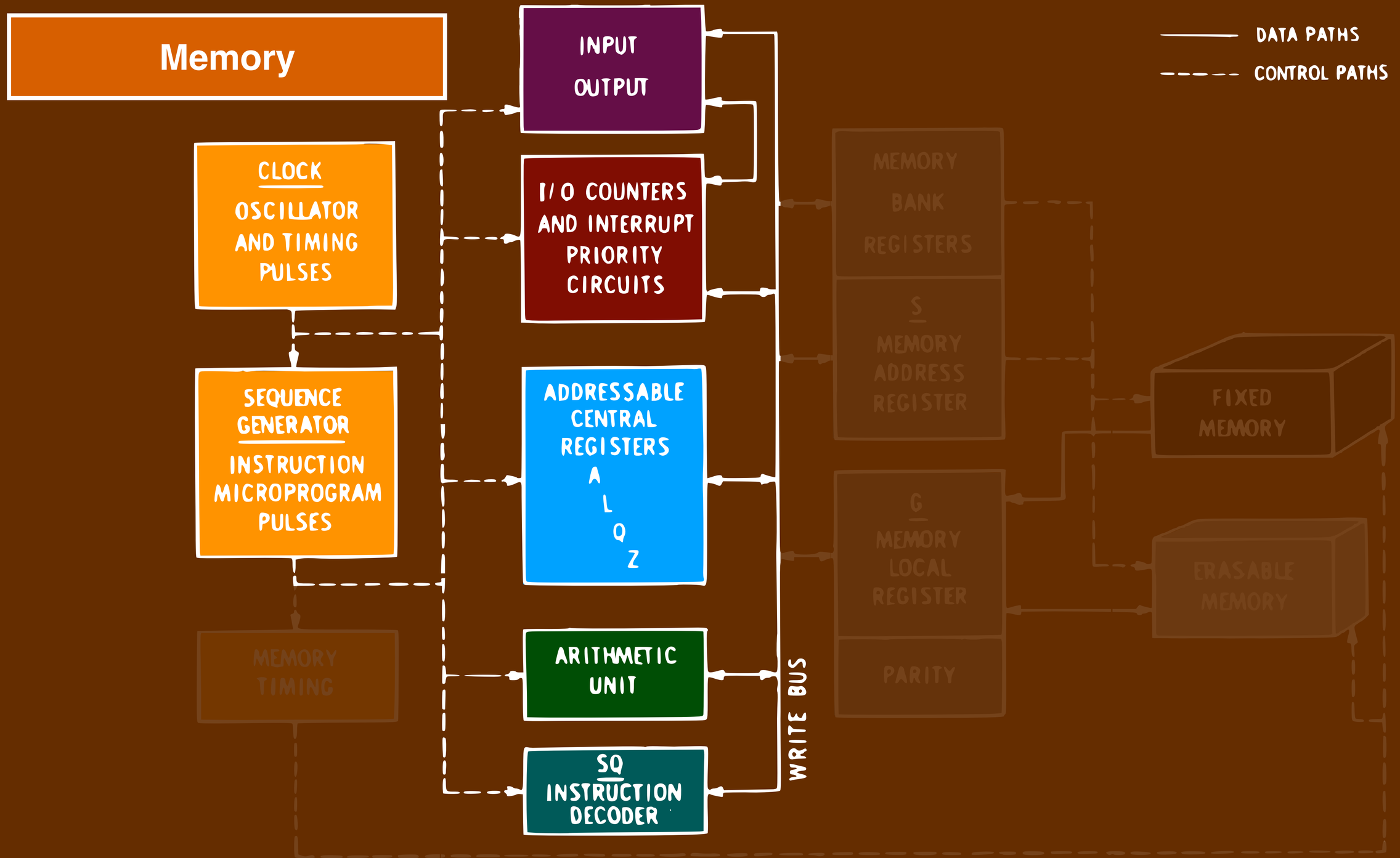

Write Bus
INPUT
OUTPUT
CLOCK
OSCILLATOR AND TIMING PULSES
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
MEMORY TIMING
ARITHMETIC UNIT
SQ
INSTRUCTION DECODER
WRITE BUS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
G
MEMORY LOCAL REGISTER
PARITY
FIXED MEMORY
ERASABLE MEMORY
DATA PATHS
CONTROL PATHS

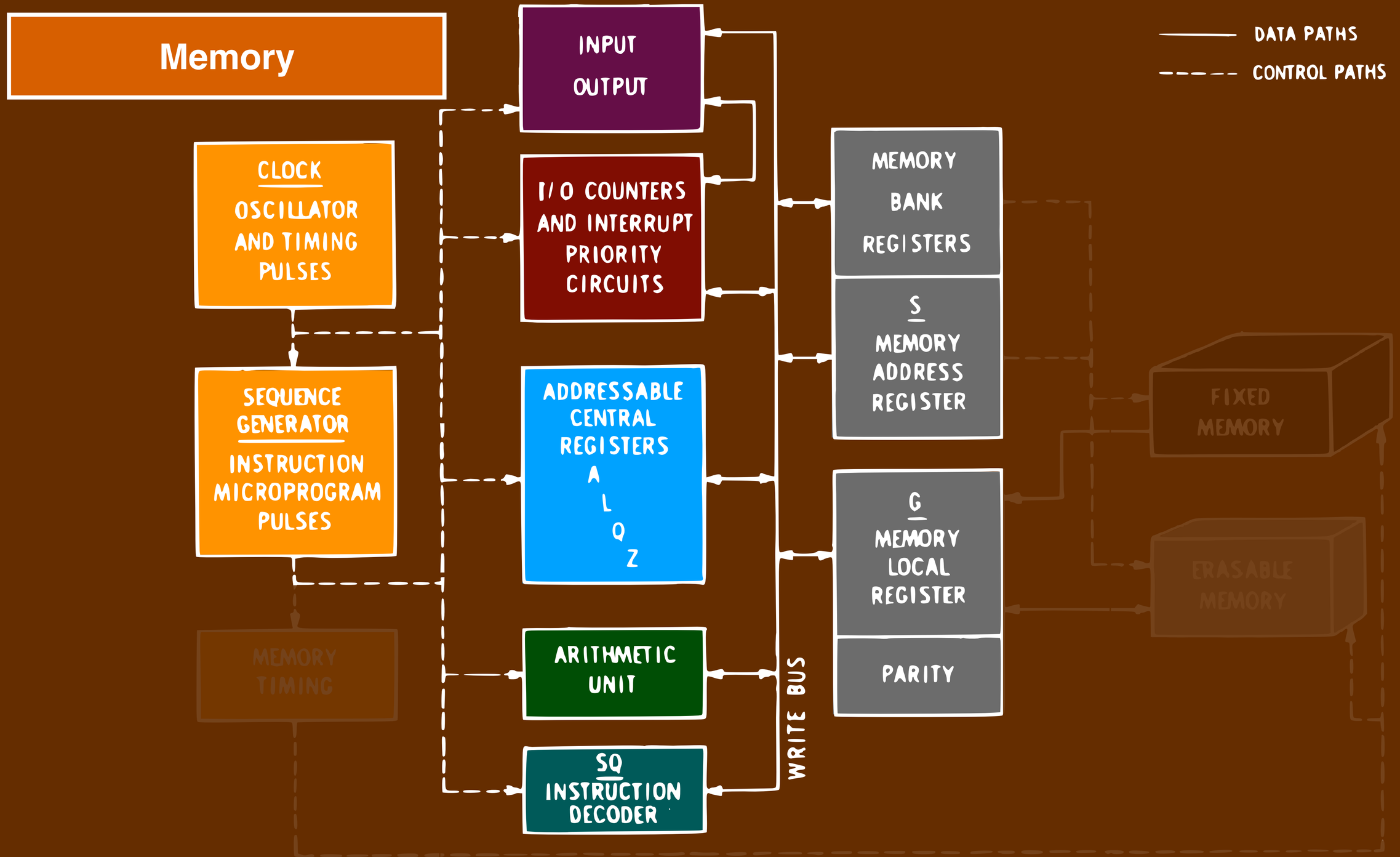

Write Bus
INPUT
OUTPUT
CLOCK
OSCILLATOR AND TIMING PULSES
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
FIXED MEMORY
G
MEMORY LOCAL REGISTER
ERASABLE MEMORY
PARITY
MEMORY TIMING
ARITHMETIC UNIT
SQ
INSTRUCTION DECODER
WRITE BUS
DATA PATHS
CONTROL PATHS

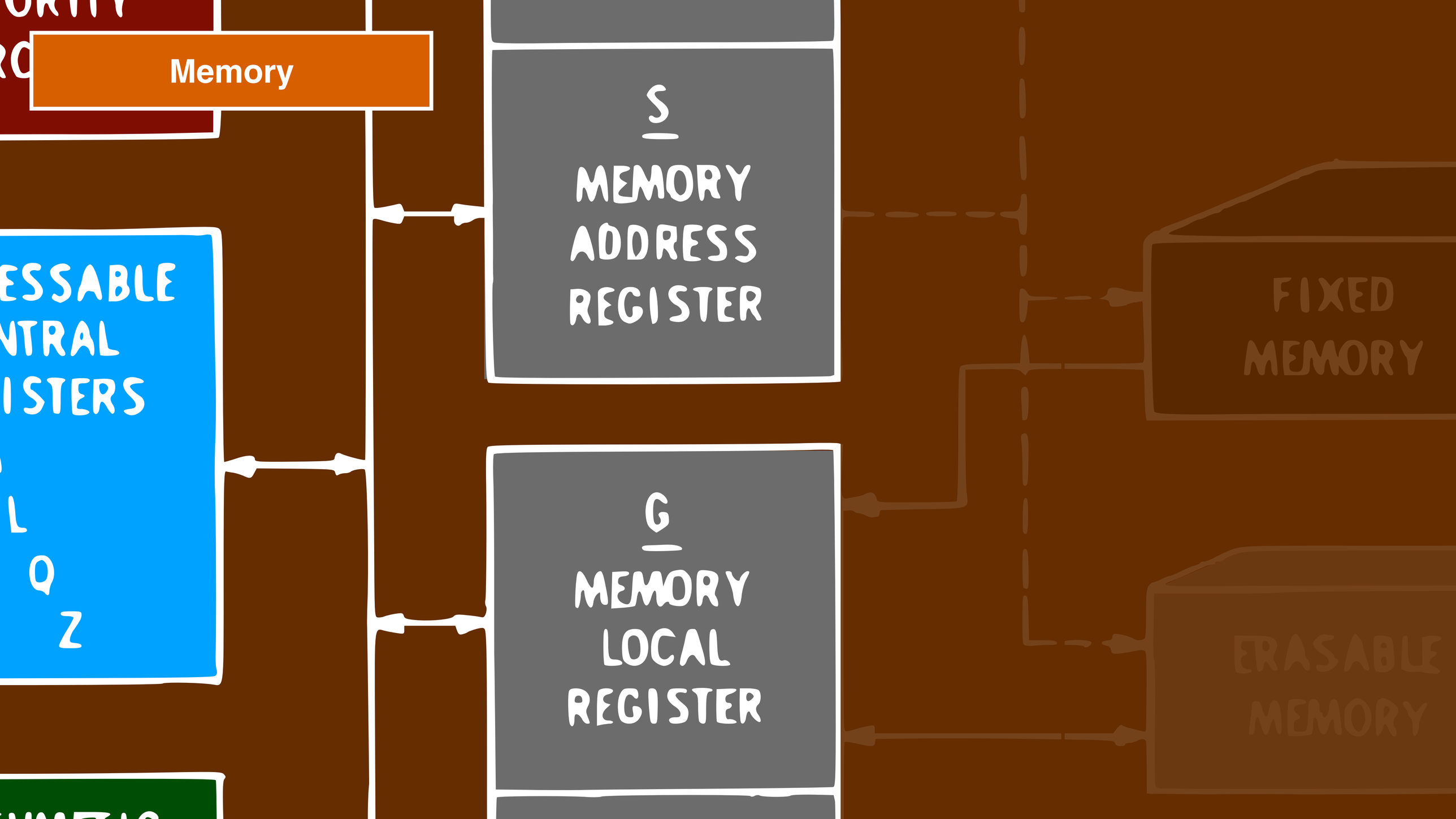

Registers
ADDRESSABLE
CENTRAL
REGISTERS
A
L
Q
Z

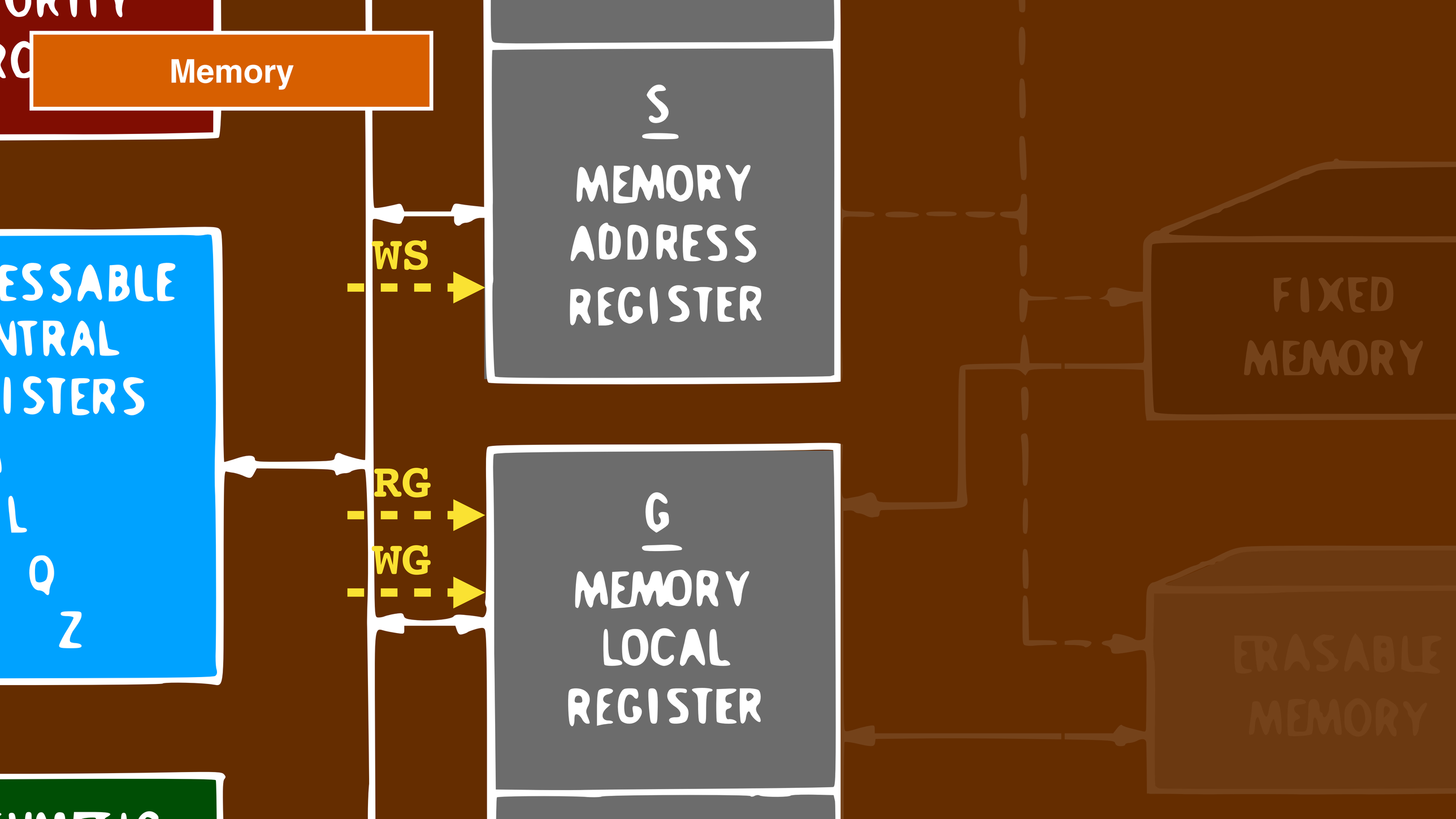

Registers
RA
ADDRESSABLE
CENTRAL
REGISTERS
A
L
Q
Z

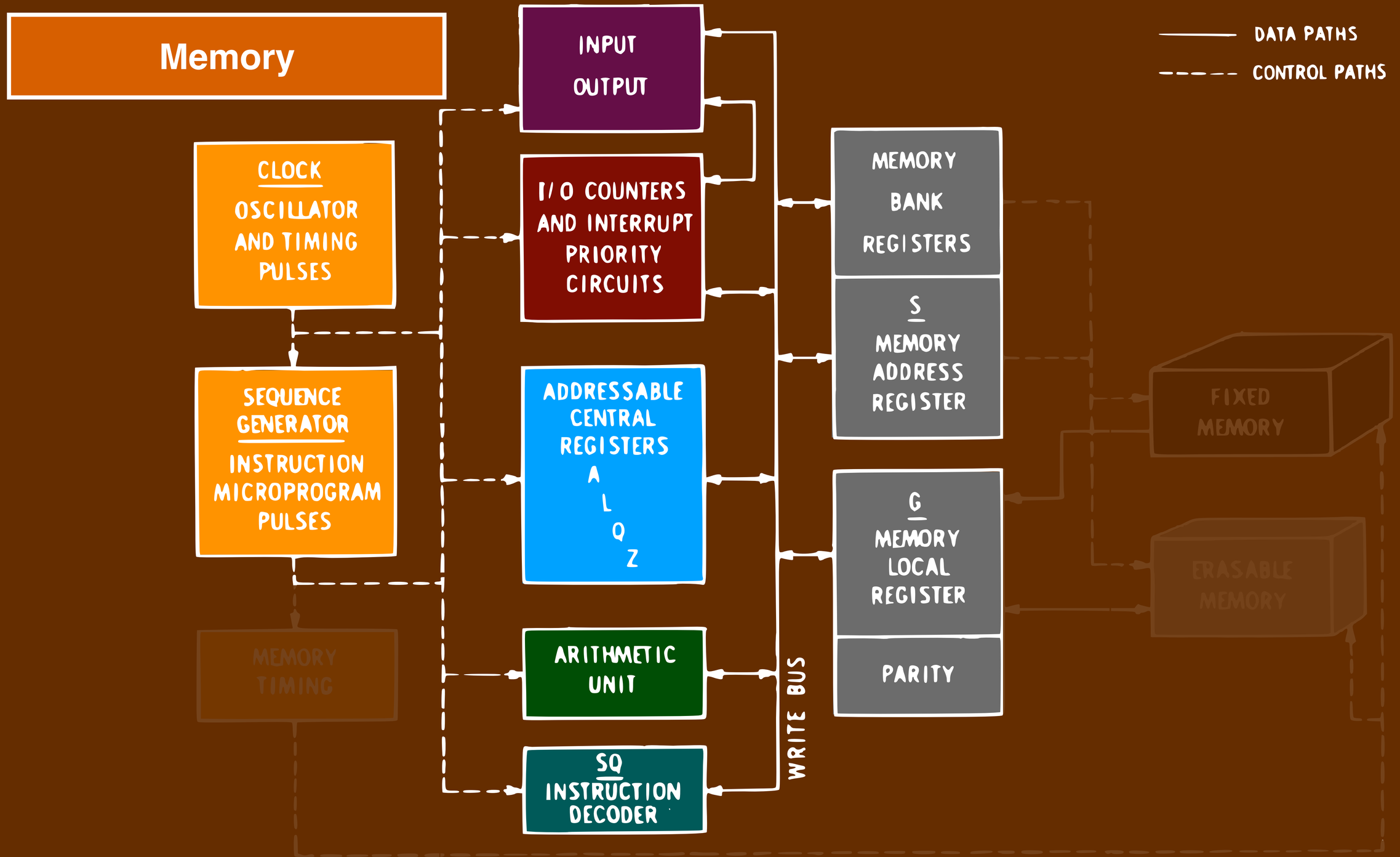

Registers
WA
ADDRESSABLE
CENTRAL
REGISTERS
A
L
Q
Z

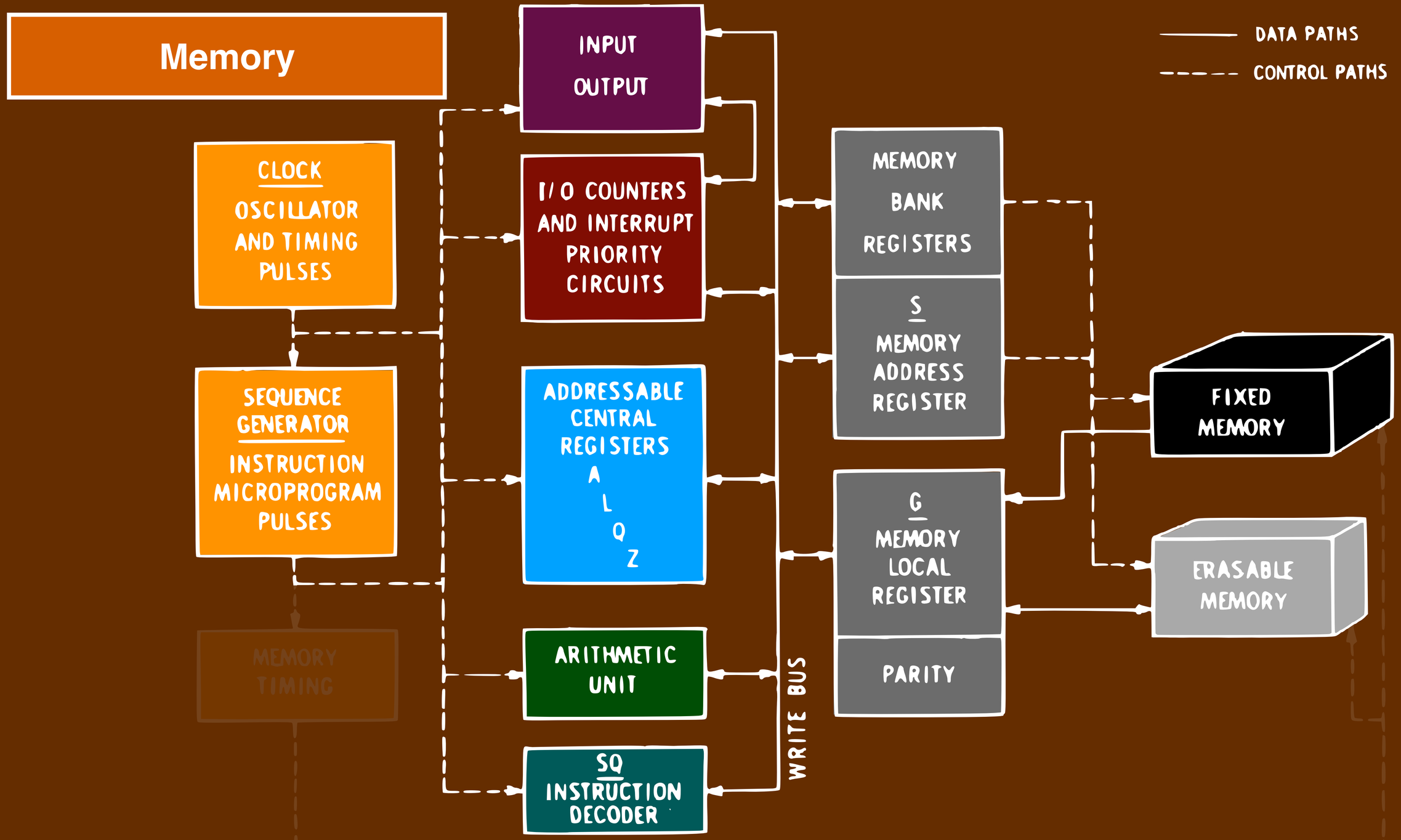

Memory
INPUT
OUTPUT
CLOCK
OSCILLATOR AND TIMING PULSES
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
MEMORY TIMING
ARITHMETIC UNIT
SQ
INSTRUCTION DECODER
WRITE BUS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
G
MEMORY LOCAL REGISTER
PARITY
FIXED MEMORY
ERASABLE MEMORY
DATA PATHS
CONTROL PATHS


Memory
INPUT
OUTPUT
CLOCK
OSCILLATOR AND TIMING PULSES
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
FIXED MEMORY
G
MEMORY LOCAL REGISTER
ERASABLE MEMORY
MEMORY TIMING
ARITHMETIC UNIT
PARITY
WRITE BUS
SQ
INSTRUCTION DECODER
DATA PATHS
CONTROL PATHS


Memory
S
MEMORY
ADDRESS
REGISTER
G
MEMORY
LOCAL
REGISTER
FIXED
MEMORY
ERASABLE
MEMORY


Memory
S
MEMORY
ADDRESS
REGISTER
WS
RG
WG
G
MEMORY
LOCAL
REGISTER
FIXED
MEMORY
ERASABLE
MEMORY

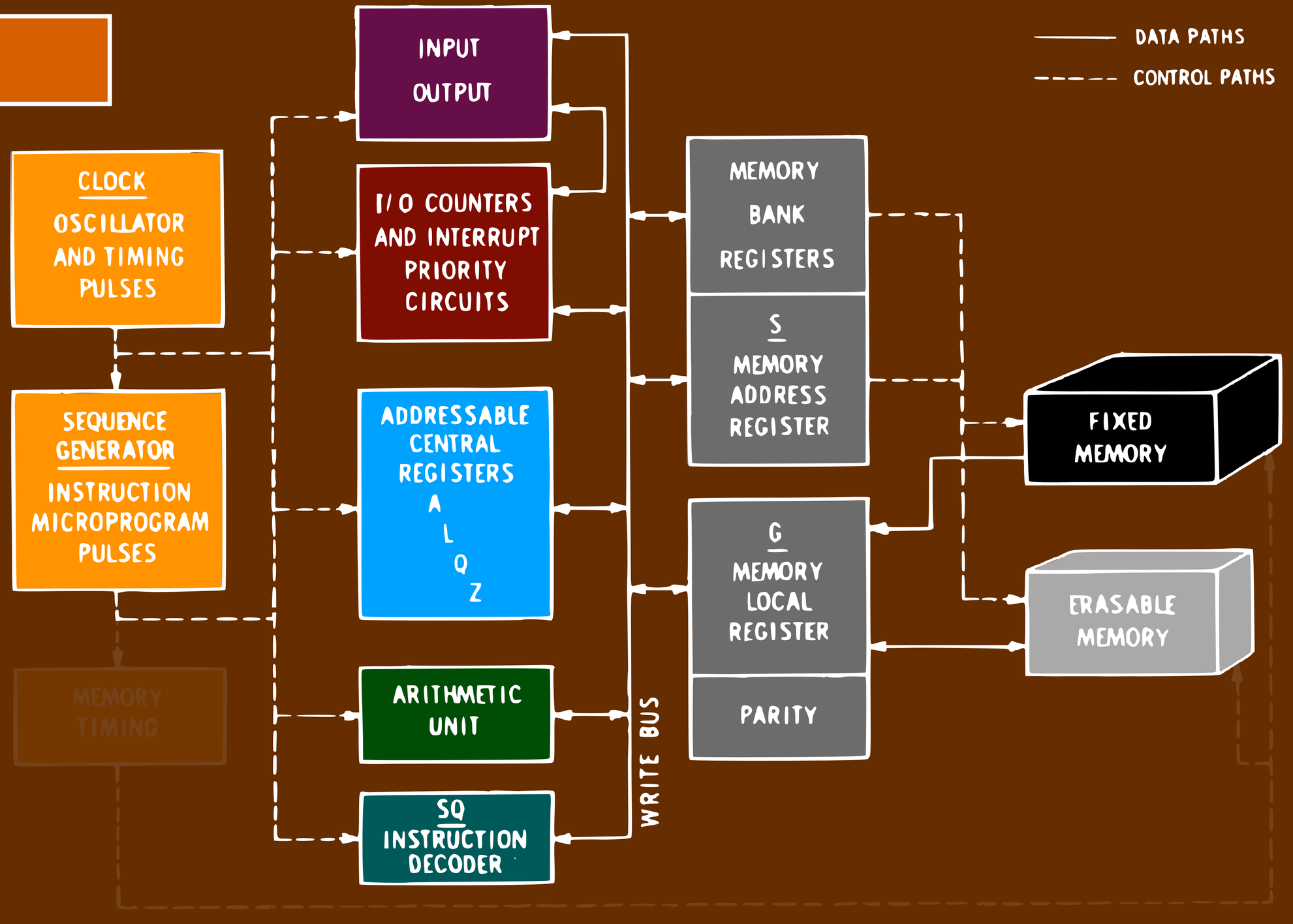

Memory
INPUT
OUTPUT
CLOCK
OSCILLATOR AND TIMING PULSES
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
FIXED MEMORY
G
MEMORY LOCAL REGISTER
ERASABLE MEMORY
MEMORY TIMING
ARITHMETIC UNIT
PARITY
WRITE BUS
SQ
INSTRUCTION DECODER
DATA PATHS
CONTROL PATHS


Memory
INPUT
OUTPUT
CLOCK
OSCILLATOR AND TIMING PULSES
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
FIXED MEMORY
G
MEMORY LOCAL REGISTER
ERASABLE MEMORY
PARITY
MEMORY TIMING
ARITHMETIC UNIT
SQ
INSTRUCTION DECODER
WRITE BUS
DATA PATHS
CONTROL PATHS

Microcode

ld a, [k]

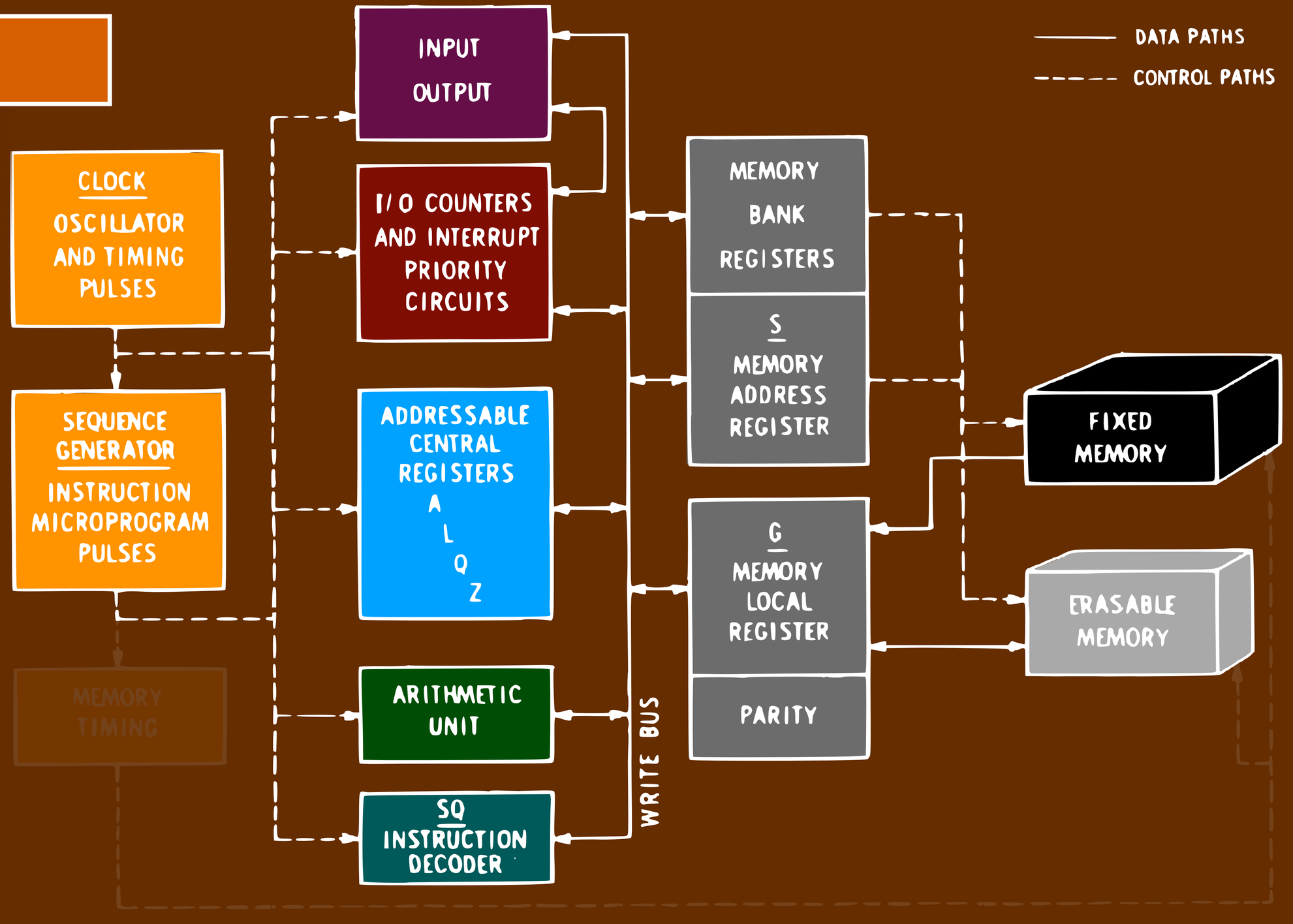
T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		
T11				
T12				
T1				
T2				
T3				
T4				
T5				

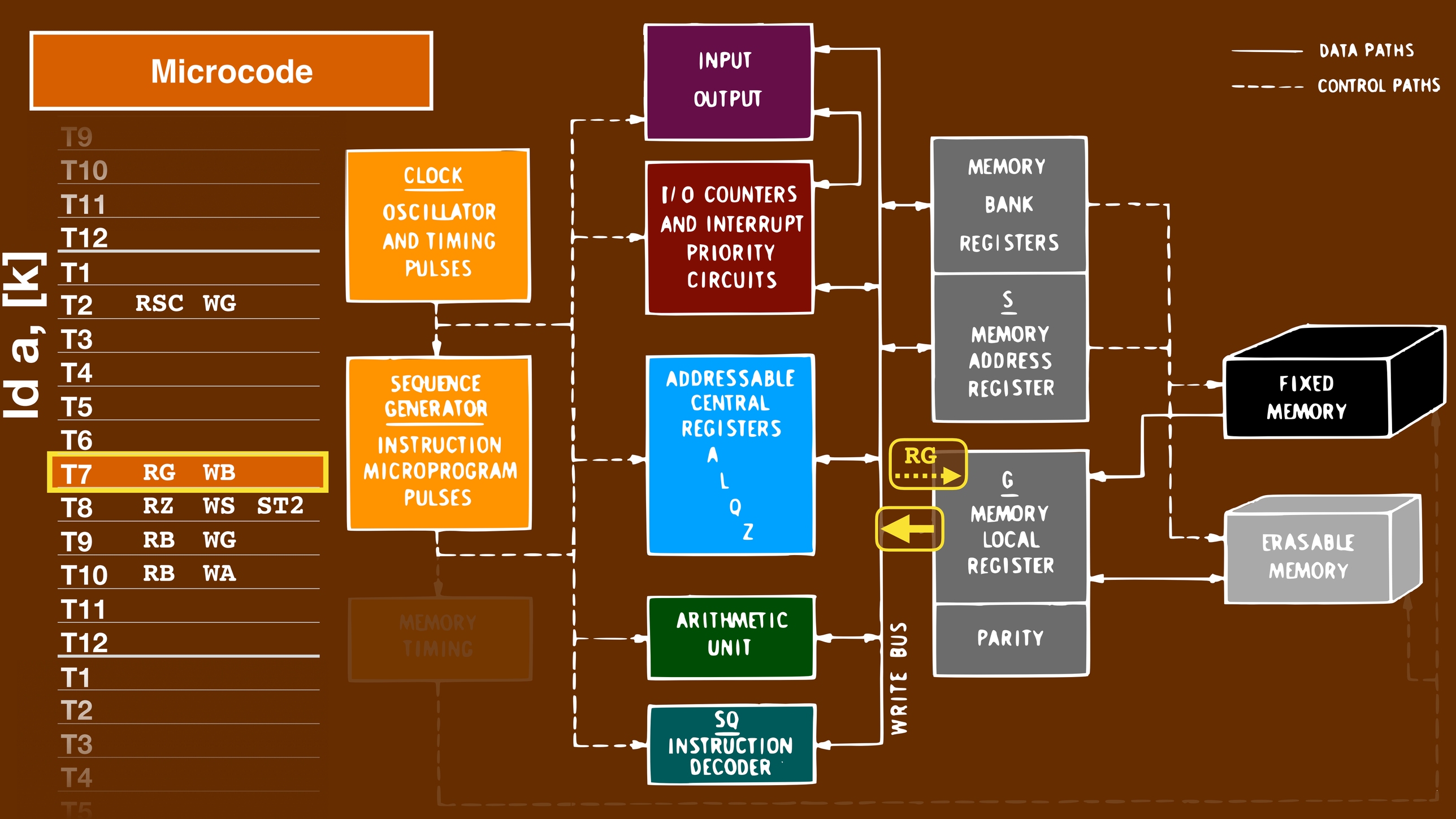


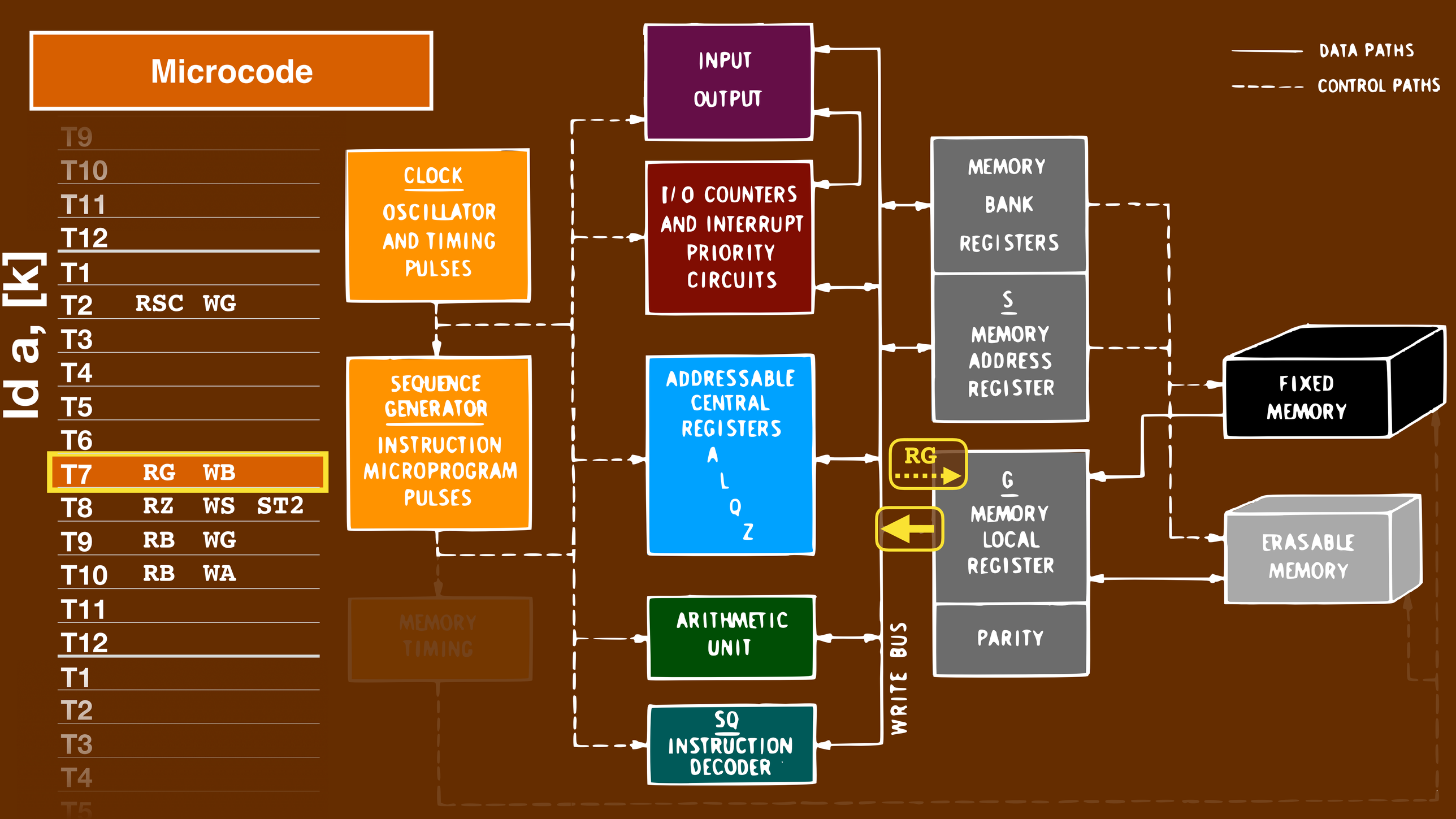
Microcode

ld a, [k]

T9			
T10			
T11			
T12			
T1			
T2	RSC	WG	
T3			
T4			
T5			
T6			
T7	RG	WB	
T8	RZ	WS	ST2
T9	RB	WG	
T10	RB	WA	
T11			
T12			
T1			
T2			
T3			
T4			
T5			



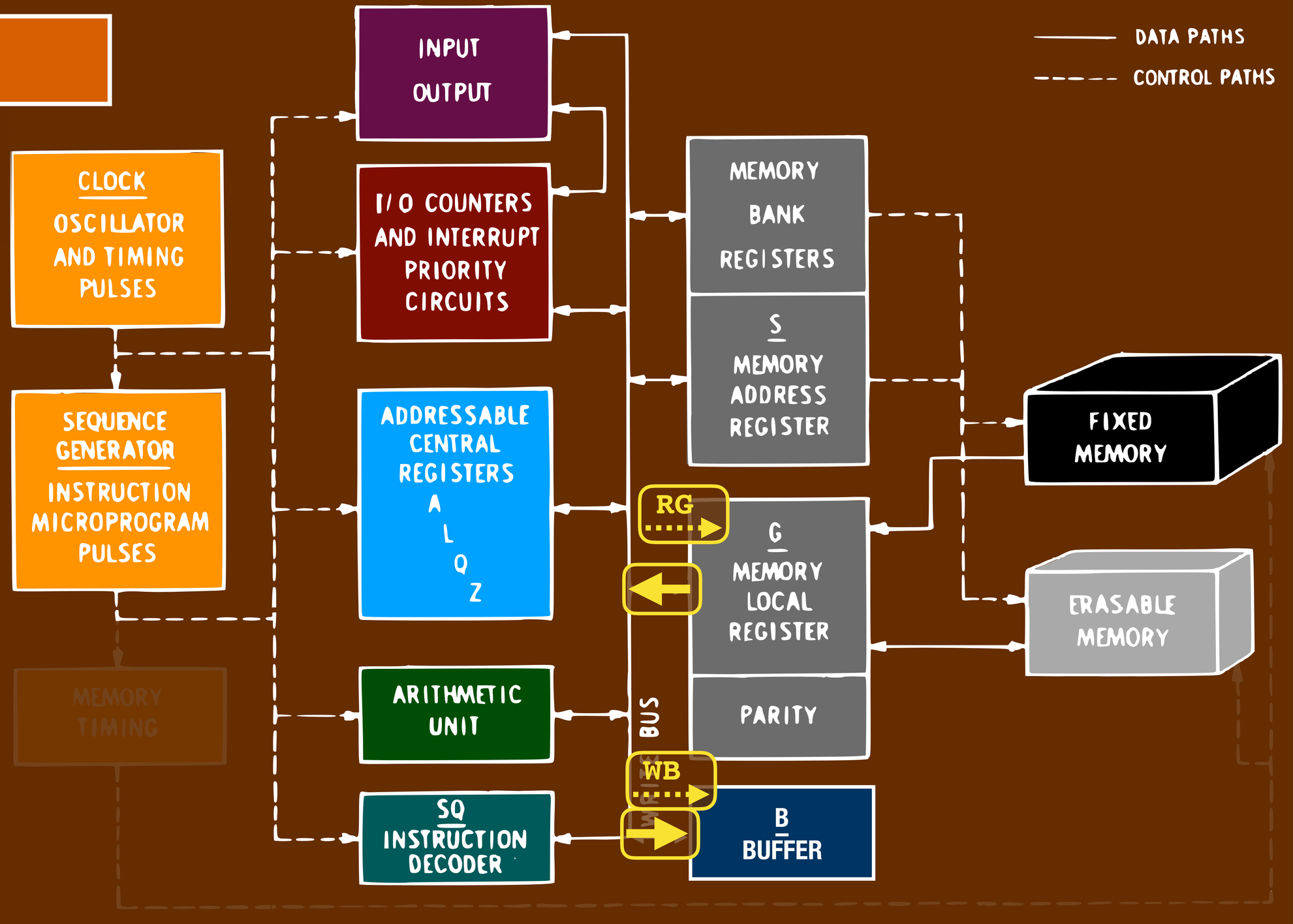

Microcode
ld a, [k]
T9
T10
T11
T12
T1
T2 RSC WG
T3
T4
T5
T6
T7 RG WB
T8 RZ WS ST2
T9 RB WG
T10 RB WA
T11
T12
T1
T2
T3
T4
T5
CLOCK
OSCILLATOR AND TIMING PULSES
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
MEMORY TIMING
INPUT
OUTPUT
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
ARITHMETIC UNIT
SQ
INSTRUCTION DECODER
WRITE BUS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
RG
G
MEMORY LOCAL REGISTER
PARITY
FIXED MEMORY
ERASABLE MEMORY
DATA PATHS
CONTROL PATHS


Microcode
ld a, [k]
T9
T10
T11
T12
T1
T2 RSC WG
T3
T4
T5
T6
T7 RG WB
T8 RZ WS ST2
T9 RB WG
T10 RB WA
T11
T12
T1
T2
T3
T4
T5
CLOCK
OSCILLATOR AND TIMING PULSES
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
MEMORY TIMING
INPUT
OUTPUT
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
ARITHMETIC UNIT
SQ
INSTRUCTION DECODER
WRITE BUS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
RG
G
MEMORY LOCAL REGISTER
PARITY
FIXED MEMORY
ERASABLE MEMORY
DATA PATHS
CONTROL PATHS

Microcode

ld a, [k]

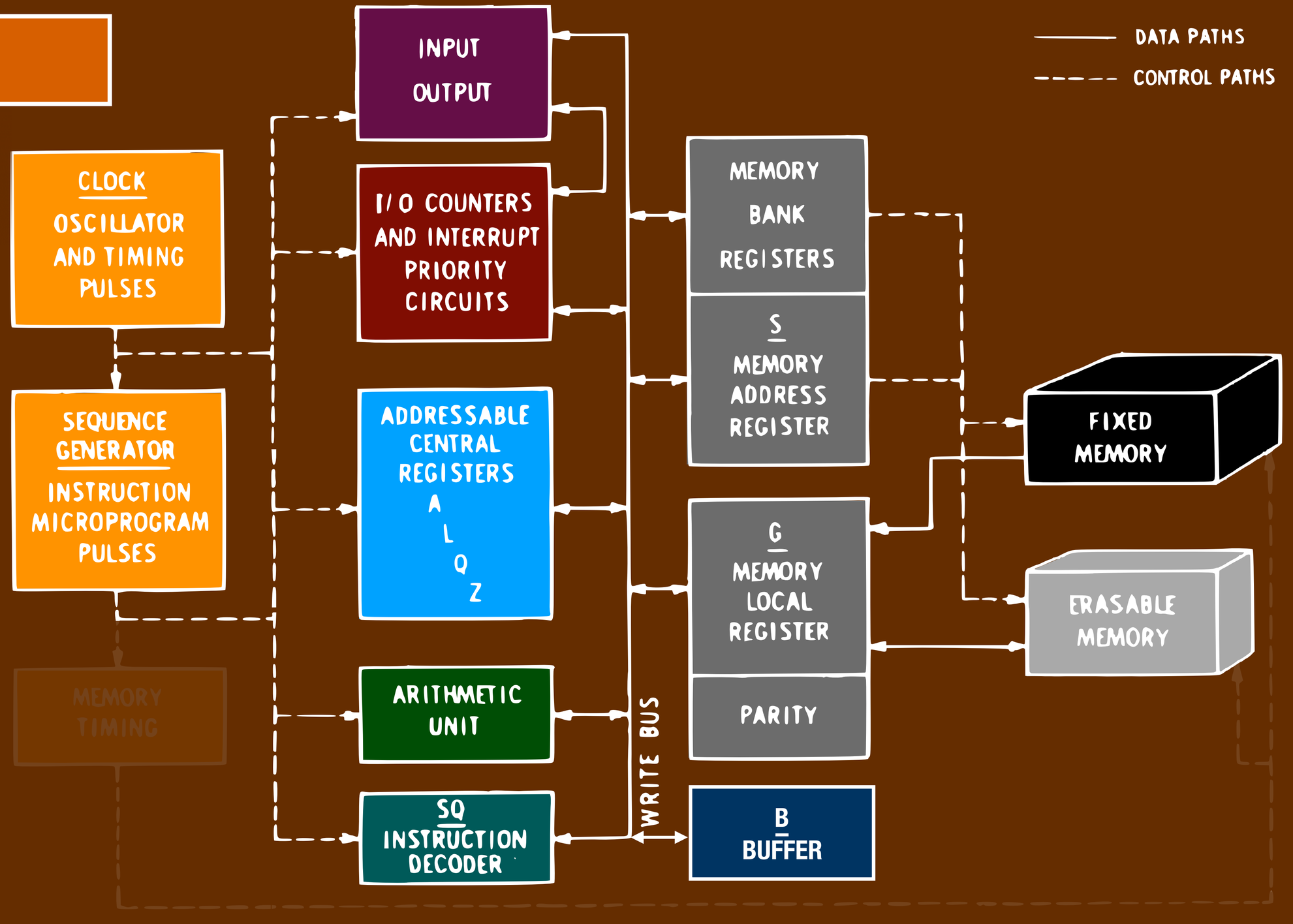
T9			
T10			
T11			
T12			
T1			
T2	RSC	WG	
T3			
T4			
T5			
T6			
T7	RG	WB	
T8	RZ	WS	ST2
T9	RB	WG	
T10	RB	WA	
T11			
T12			
T1			
T2			
T3			
T4			
T5			



Microcode

ld a, [k]

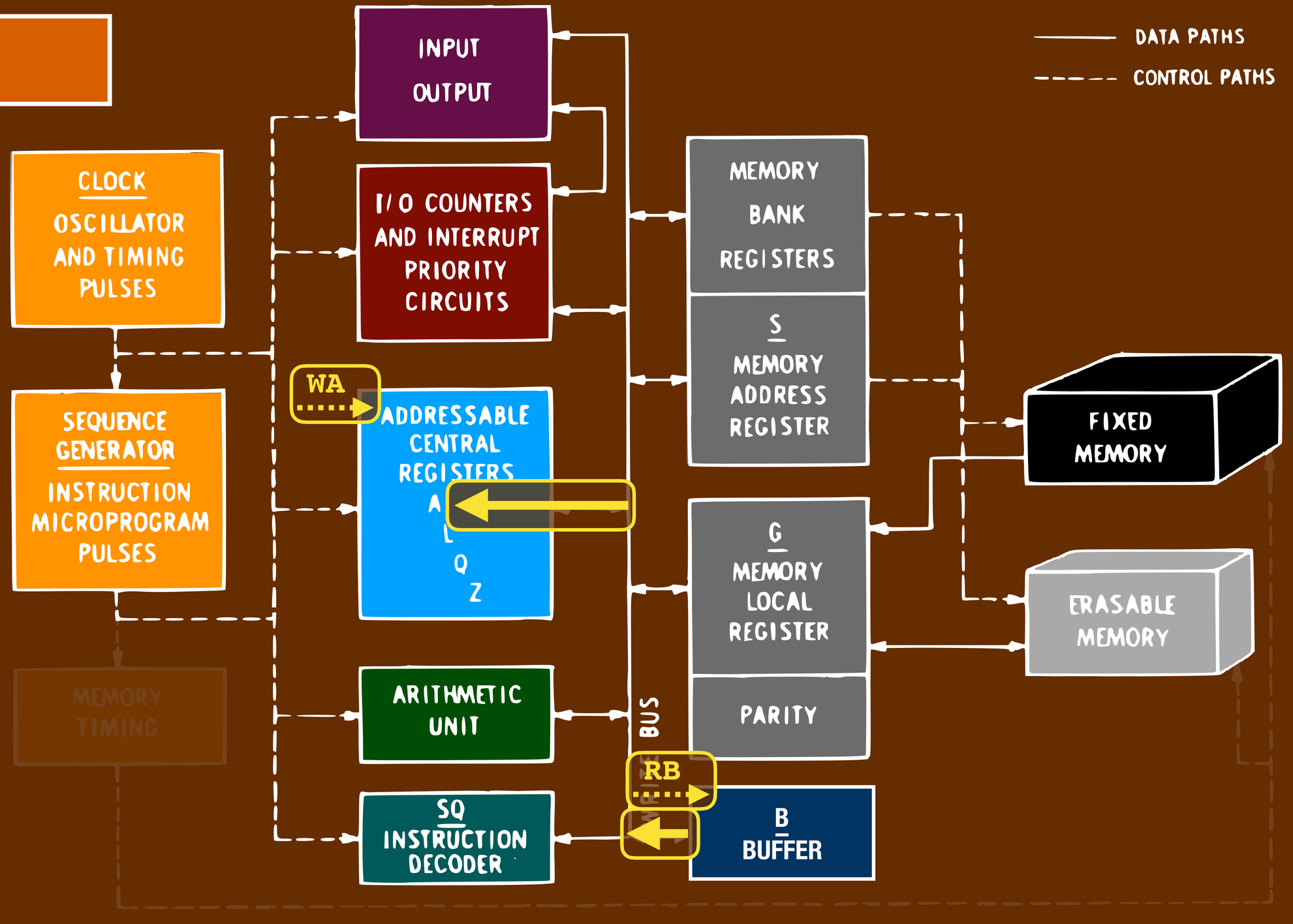
T9			
T10			
T11			
T12			
T1			
T2	RSC	WG	
T3			
T4			
T5			
T6			
T7	RG	WB	
T8	RZ	WS	ST2
T9	RB	WG	
T10	RB	WA	
T11			
T12			
T1			
T2			
T3			
T4			
T5			



Microcode

ld a, [k]

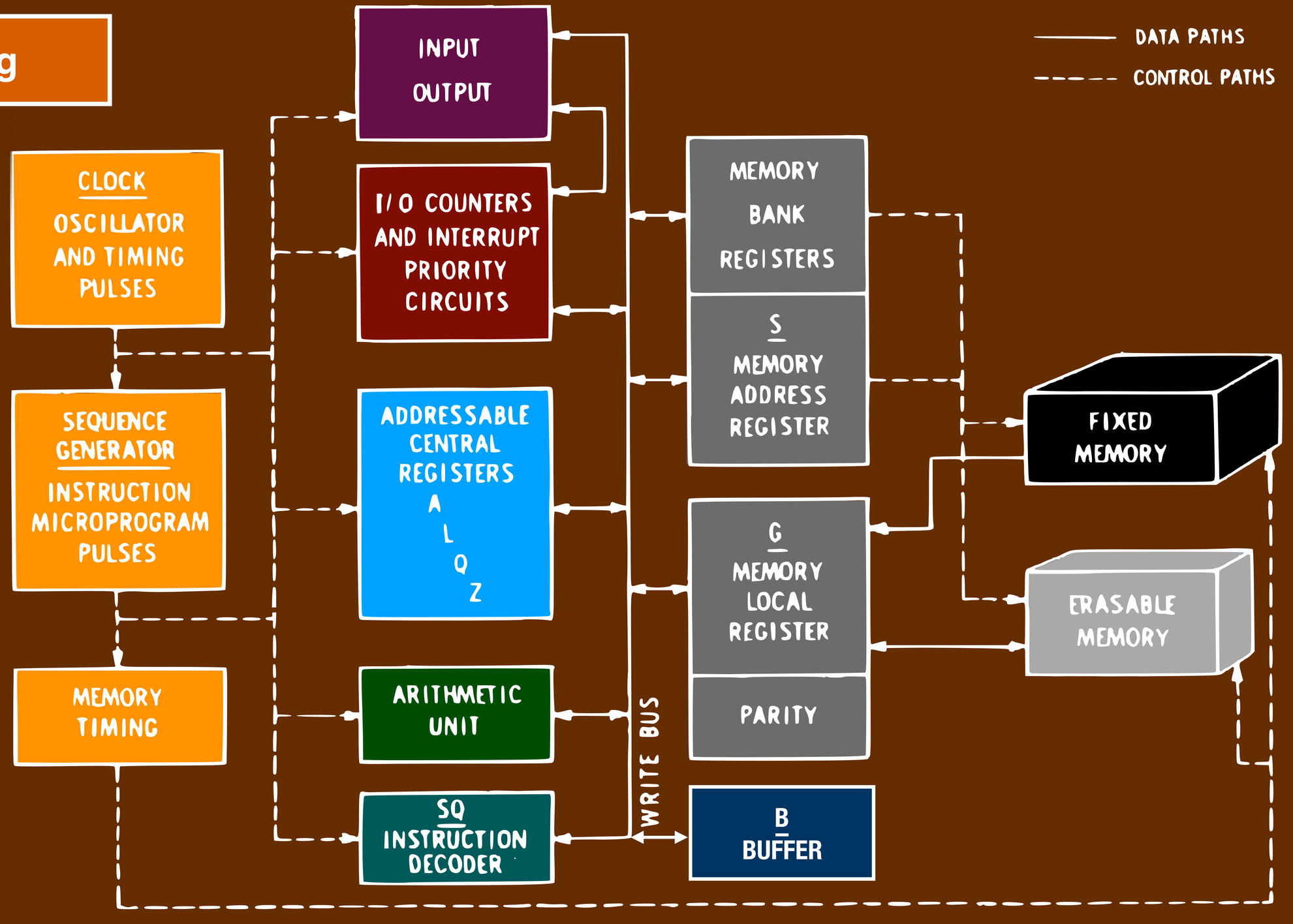
T9			
T10			
T11			
T12			
T1			
T2	RSC	WG	
T3			
T4			
T5			
T6			
T7	RG	WB	
T8	RZ	WS	ST2
T9	RB	WG	
T10	RB	WA	
T11			
T12			
T1			
T2			
T3			
T4			
T5			



Memory Timing

ld a, [k]

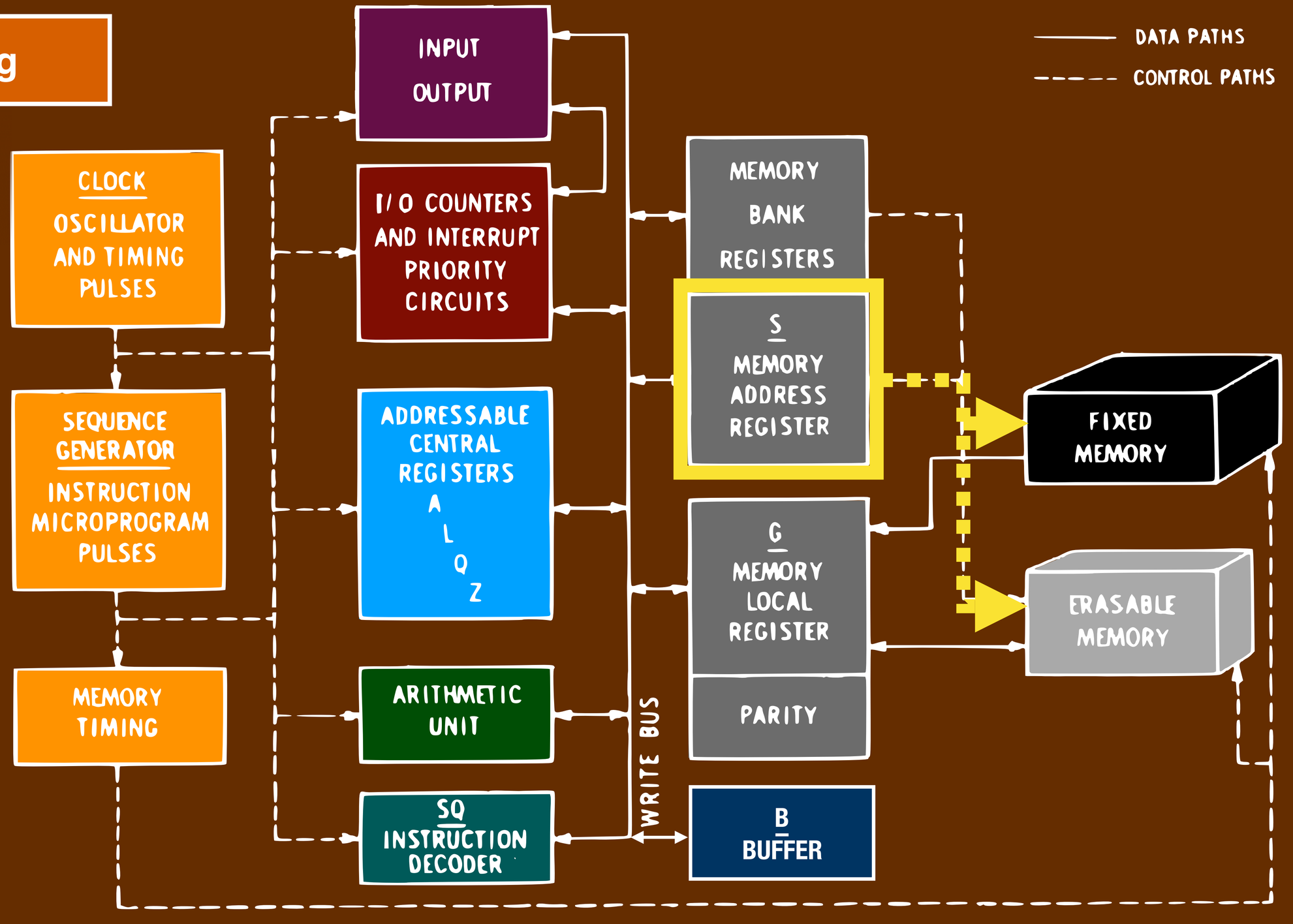
T9			
T10			
T11			
T12			
T1			
T2	RSC	WG	
T3			
T4			
T5			
T6			
T7	RG	WB	
T8	RZ	WS	ST2
T9	RB	WG	
T10	RB	WA	
T11			
T12			
T1			
T2			
T3			
T4			
T5			



Memory Timing

ld a, [k]

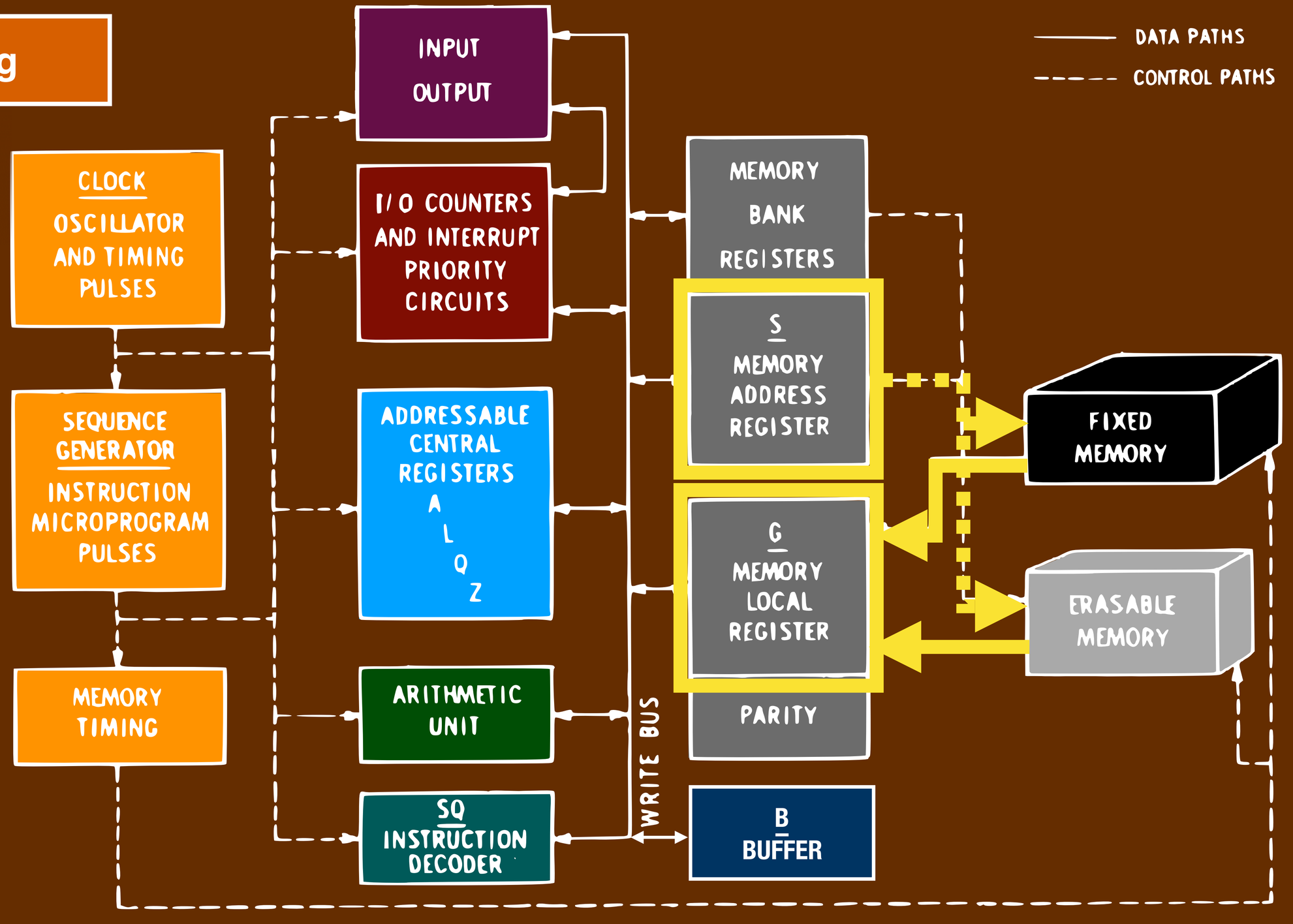
T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		READ G = M[S]
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		
T11				
T12				
T1				
T2				
T3				
T4				
T5				



Memory Timing

ld a, [k]

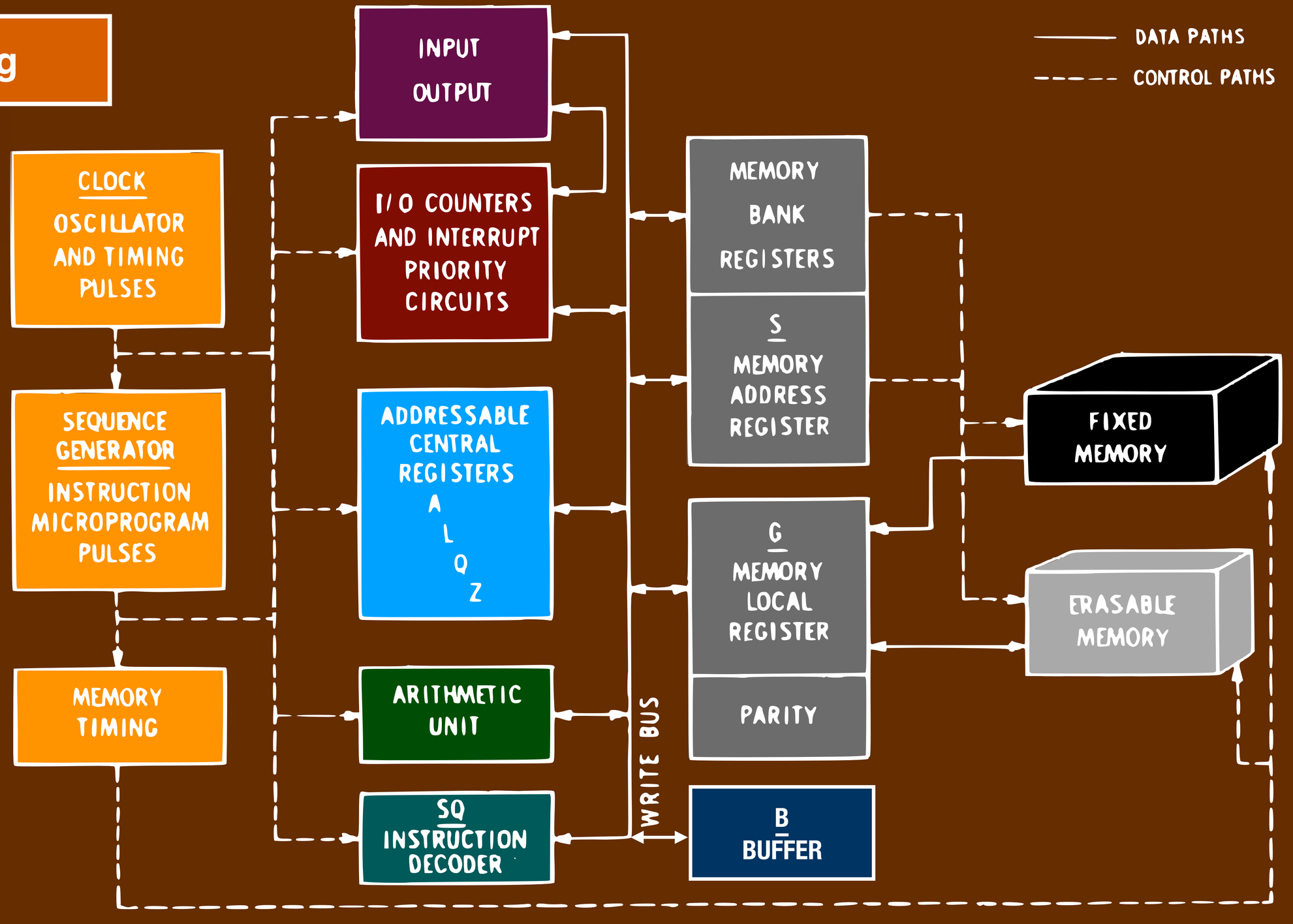
T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		READ G = M[S]
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		
T11				
T12				
T1				
T2				
T3				
T4				
T5				



Memory Timing

ld a, [k]

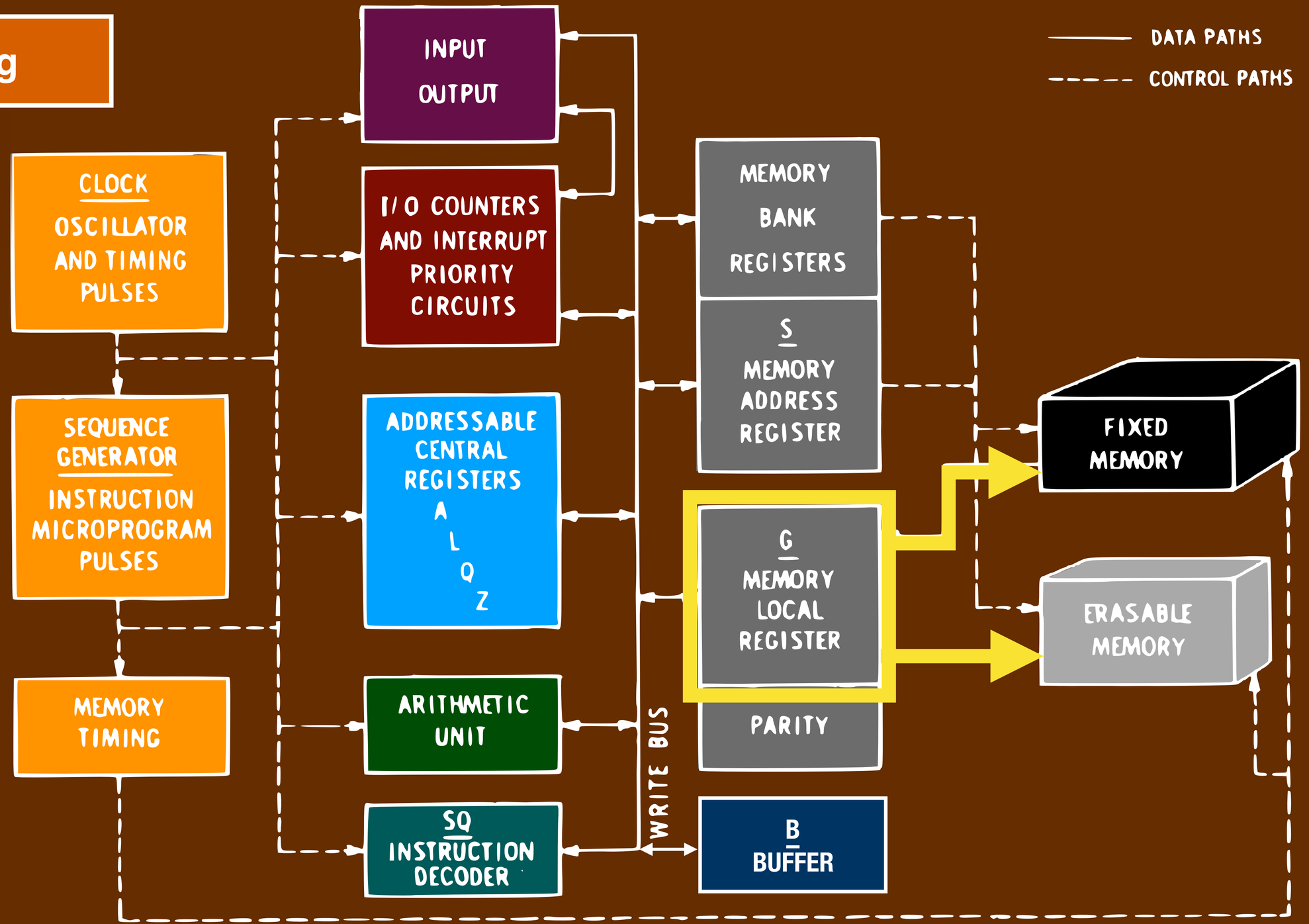
T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		READ G = M[S]
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		
T11				
T12				
T1				
T2				
T3				
T4				
T5				



Memory Timing

ld a, [k]

T9			
T10			
T11			
T12			
T1			
T2	RSC	WG	READ G = M[S]
T3			
T4			
T5			
T6			
T7	RG	WB	
T8	RZ	WS	ST2
T9	RB	WG	
T10	RB	WA	WRITE M[S] = G
T11			
T12			
T1			
T2			
T3			
T4			
T5			



Microcode

T9			
T10			
T11			
T12			
T1			
T2	RSC	WG	READ G = M[S]
T3			
T4			
T5			
T6			
T7	RG	WB	
T8	RZ	WS	ST2
T9	RB	WG	
T10	RB	WA	WRITE M[S] = G
T11			
T12			
T1			
T2			
T3			
T4			
T5			

Microcode

Memory Timing

G = M[S]

M[S_{old}] = G

T9			
T10			
T11			
T12			
T1			
T2	RSC	WG	
T3			
T4			
T5			
T6			
T7	RG	WB	
T8	RZ	WS	ST2
T9	RB	WG	
T10	RB	WA	
T11			
T12			
T1			
T2			
T3			
T4			
T5			

Microcode

Memory Timing

T9	
T10	
T11	
T12	
T1	
T2	RSC WG
T3	
T4	
T5	
T6	
T7	RG WB
T8	RZ WS ST2
T9	RB WG
T10	RB WA
T11	
T12	
T1	
T2	
T3	
T4	
T5	

ld a, [k]

`G = r[S]`

`B = G`
`S = Z`
`G = B`
`A = B`

`G = M[S]`

`M[S_old] = G`

Microcode

Memory Timing

`G = M[S]`

`M[S_old] = G`

T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		`G = r[S]`
T3				
T4				
T5				
T6				
T7	RG	WB		`B = G`
T8	RZ	WS	ST2	`S = Z`
T9	RB	WG		`G = B`
T10	RB	WA		`A = B`
T11				
T12				
T1				
T2				
T3				
T4				
T5				

ld a, [k]

Microcode

Memory Timing

G = M[S]

M[S_{old}] = G

T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		
T11				
T12				
T1				
T2				
T3				
T4				
T5				

ld a, [k]

G = r[S]

B = G
S = Z
G = B
A = B

Microcode

Memory Timing

`G = M[S]`

`M[S_old] = G`

T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		
T11				
T12				
T1				
T2				
T3				
T4				
T5				

ld a, [k]

`G = r[S]`

`B = G`

`S = Z`

`G = B`

`A = B`

Microcode

Memory Timing

```
G = M[S]

M[S_old] = G
```

```
T9
T10
T11
T12
T1
T2    RSC  WG
T3
T4
T5
T6
T7    RG   WB
T8    RZ   WS   ST2
T9    RB   WG
T10   RB   WA
T11
T12
T1
T2
T3
T4
```

READ: G = M[S]

WRITE: M[S] = G

ld a, [k]

```
G = r[S]

B = G
S = Z
G = B
A = B
```

Microcode

Memory Timing

```
G = M[S]

M[S_old] = G
```

ld a, [k]

T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		READ G = M[S]
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		WRITE M[S] = G
T11				
T12				
T1				
T2				
T3				
T4				

```
G = r[S]

B = G
S = Z
G = B
A = B
```

xchg a, [k]

T9			
T10			
T11			
T12			
T1	RL10BB	WS	
T2	RSC	WG	
T3	RA	WB	
T4			
T5	RG	WA	
T6			
T7	RB	WSC	WG
T8	RZ	WS	ST2
T9			
T10			
T11			
T12			
T1			
T2			
T3			
T4			

```
S = B[0:9]
G = r[S]
B = A

A = G

r[S] = B, G = B
S = Z
```

Microcode

Memory Timing

```
G = M[S]

M[Sold] = G
```

ld a, [k]

T9				
T10				
T11				
T12				
T1				
T2	RSC	WG	READ G = M[S]	G = r[S]
T3				
T4				
T5				
T6				
T7	RG	WB		B = G
T8	RZ	WS	ST2	S = Z
T9	RB	WG		G = B
T10	RB	WA	WRITE M[S] = G	A = B
T11				
T12				
T1				
T2				
T3				
T4				

xchg a, [k]

T9				
T10				
T11				
T12				
T1	RL10BB	WS		S = B[0:9]
T2	RSC	WG		G = r[S]
T3	RA	WB		B = A
T4				
T5	RG	WA		A = G
T6				
T7	RB	WSC	WG	r[S] = B, G = B
T8	RZ	WS	ST2	S = Z
T9				
T10				
T11				
T12				
T1				
T2				
T3				
T4				

Microcode

Memory Timing

G = M[S]

M[S$_{old}$] = G

ld a, [k]

T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		READ G = M[S]
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		WRITE M[S] = G
T11				
T12				
T1				
T2				
T3				
T4				

G = r[S]

B = G

S = Z

G = B

A = B

xchg a, [k]

T9			
T10			
T11			
T12			
T1	RL10BB	WS	
T2	RSC	WG	
T3	RA	WB	
T4			
T5	RG	WA	
T6			
T7	RB	WSC	WG
T8	RZ	WS	ST2
T9			
T10			
T11			
T12			
T1			
T2			
T3			
T4			

S = B[0:9]

G = r[S]

B = A

A = G

r[S] = B, G = B

S = Z

Microcode

Memory Timing

`G = M[S]`

`M[S_old] = G`

ld a, [k]

T9				
T10				
T11				
T12				
T1				
T2	RSC	WG	READ	G = M[S]
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA	WRITE	M[S] = G
T11				
T12				
T1				
T2				
T3				
T4				

`G = r[S]`

`B = G`
`S = Z`
`G = B`
`A = B`

xchg a, [k]

T9			
T10			
T11			
T12			
T1	RL10BB	WS	
T2	RSC	WG	
T3	RA	WB	
T4			
T5	RG	WA	
T6			
T7	RB	WSC	WG
T8	RZ	WS	ST2
T9			
T10			
T11			
T12			
T1			
T2			
T3			
T4			

`S = B[0:9]`
`G = r[S]`
`B = A`

`A = G`

`r[S] = B, G = B`
`S = Z`

Microcode

Memory Timing

`G = M[S]`

`M[S_old] = G`

ld a, [k]

T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		READ G = M[S]
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		WRITE M[S] = G
T11				
T12				
T1				
T2				
T3				
T4				

`G = r[S]`

`B = G`
`S = Z`
`G = B`
`A = B`

xchg a, [k]

T9			
T10			
T11			
T12			
T1	RL10BB	WS	
T2	RSC	WG	
T3	RA	WB	
T4			
T5	RG	WA	
T6			
T7	RB	WSC	WG
T8	RZ	WS	ST2
T9			
T10			
T11			
T12			
T1			
T2			
T3			
T4			

`S = B[0:9]`
`G = r[S]`
`B = A`

`A = G`

`r[S] = B,` `G = B`
`S = Z`

Microcode

Memory Timing

`G = M[S]`

`M[S_old] = G`

ld a, [k]

T9				
T10				
T11				
T12				
T1				
T2	RSC	WG		READ: G = M[S]
T3				
T4				
T5				
T6				
T7	RG	WB		
T8	RZ	WS	ST2	
T9	RB	WG		
T10	RB	WA		WRITE: M[S] = G
T11				
T12				
T1				
T2				
T3				
T4				

`G = r[S]`

`B = G`
`S = Z`
`G = B`
`A = B`

xchg a, [k]

T9			
T10			
T11			
T12			
T1	RL10BB	WS	
T2	RSC	WG	
T3	RA	WB	
T4			
T5	RG	WA	
T6			
T7	RB	WSC	WG
T8	RZ	WS	ST2
T9			
T10			
T11			
T12			
T1			
T2			
T3			
T4			

`S = B[0:9]`
`G = r[S]`
`B = A`

`A = G`

`r[S] = B,` `G = B`
`S = Z`

Microcode

DV0

```
 1. xx  RA    WB    TSGN
 2. 0x  RC    WA    TMZ   DVST
 2. 1x  DVST
 3. xx  RU    WB    STAGE
```

DV1

```
 4. x0  RL    WB
 4. x1  RL    WB    TSGN
 5. 0x  RB    WY    B15X
 5. 1x  RC    WY    B15X  Z16
 6. xx  RU    WL    TOV
 7. xx  RG    RSC   WB    TSGN
 8. x0  RA    WY    PONEX
 8. x1  RA    WY
 9. 0x  RB    WA
 9. 1x  RC    WA    Z15
10. xx  RU    WB
11. xx  RL    WYD
12. xx  RU    WL
 1. xx  L2GD  RB    WYD   A2X   PIFL
 2. 0x  RG    WL    TSGU  DVST  CLXC
 2. 1x  RG    WL    TSGU  DVST  RB1F
 3. xx  RU    WB    STAGE
```

DV3

```
 4. xx  L2GD  RB    WYD   A2X   PIFL
 5. 0x  RG    WL    TSGU  DVST  CLXC
 5. 1x  RG    WL    TSGU  DVST  RB1F
 6. xx  RU    WB
 7. xx  L2GD  RB    WYD   A2X   PIFL
 8. 0x  RG    WL    TSGU  DVST  CLXC
 8. 1x  RG    WL    TSGU  DVST  RB1F
 9. xx  RU    WB
10. xx  L2GD  RB    WYD   A2X   PIFL
11. 0x  RG    WL    TSGU  DVST  CLXC
11. 1x  RG    WL    TSGU  DVST  RB1F
12. xx  RU    WB
 1. xx  L2GD  RB    WYD   A2X   PIFL
 2. 0x  RG    WL    TSGU  DVST  CLXC
 2. 1x  RG    WL    TSGU  DVST  RB1F
 3. xx  RU    WB    STAGE
```

DV4

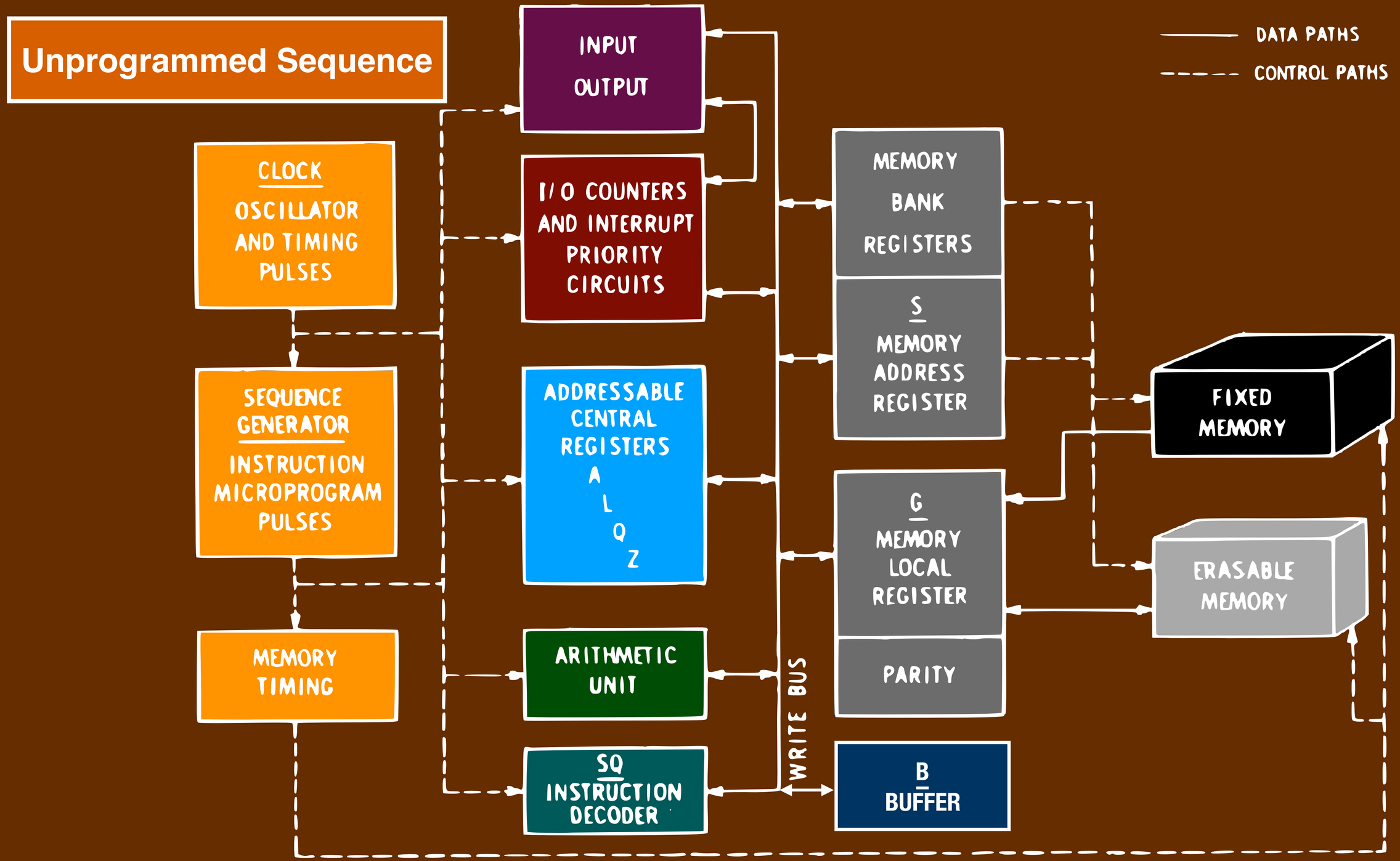
```
 3. xx  RU    WB    STAGE
 4. xx  L2GD  RB    WYD   A2X   PIFL
 5. 0x  RG    WB    TSGU  WA    CLXC
 5. 1x  RG    WB    TSGU  WA    RB1F
 6. xx  RZ    TOV
 7. 01  RC    WA
 7. 1x  RC    WA
 8. xx  RZ    WS    ST2   TSGN  RSTSTG
 9. xx  RU    WB    WL
10. 0x  RC    WL
```

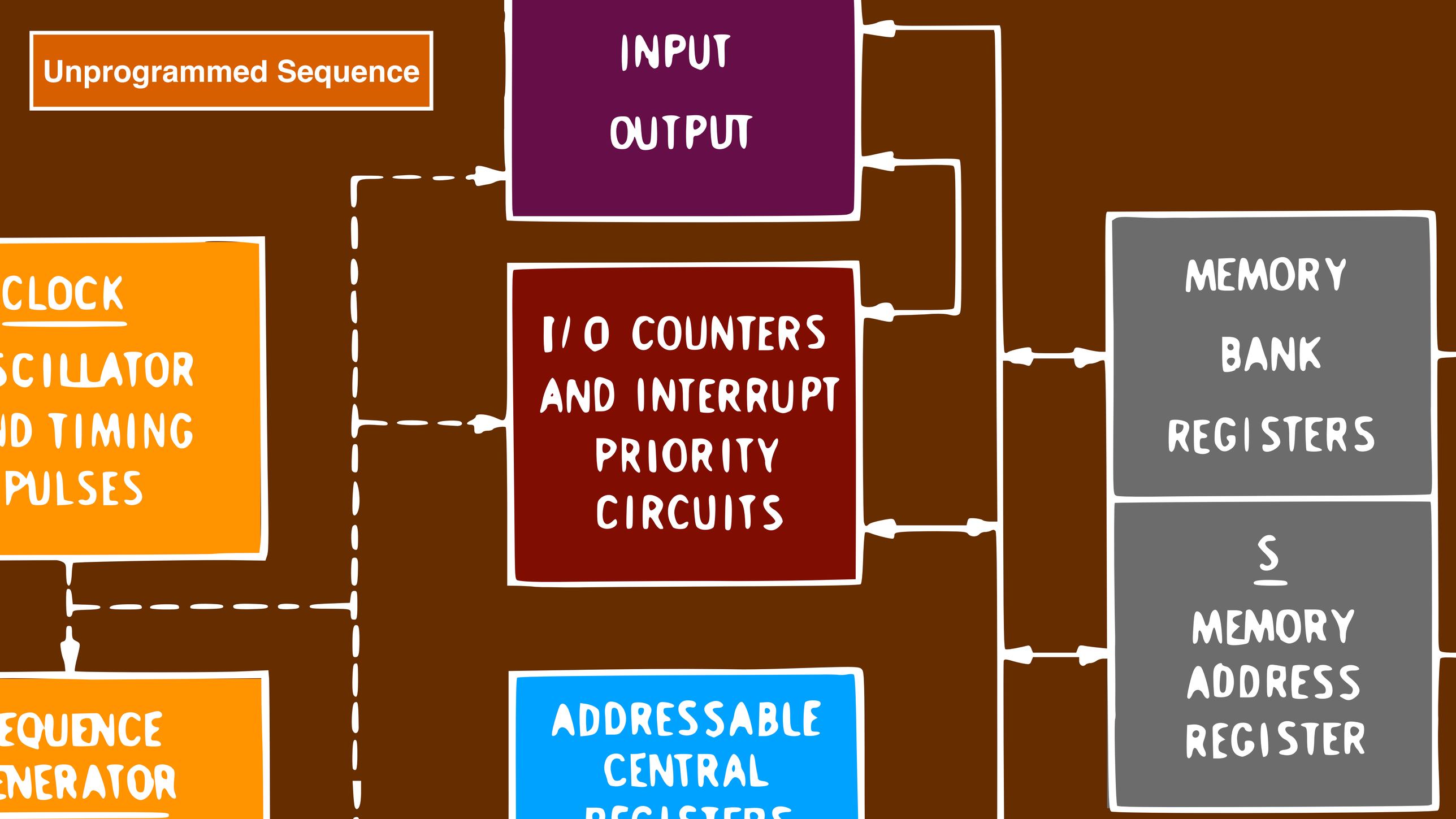
DV6

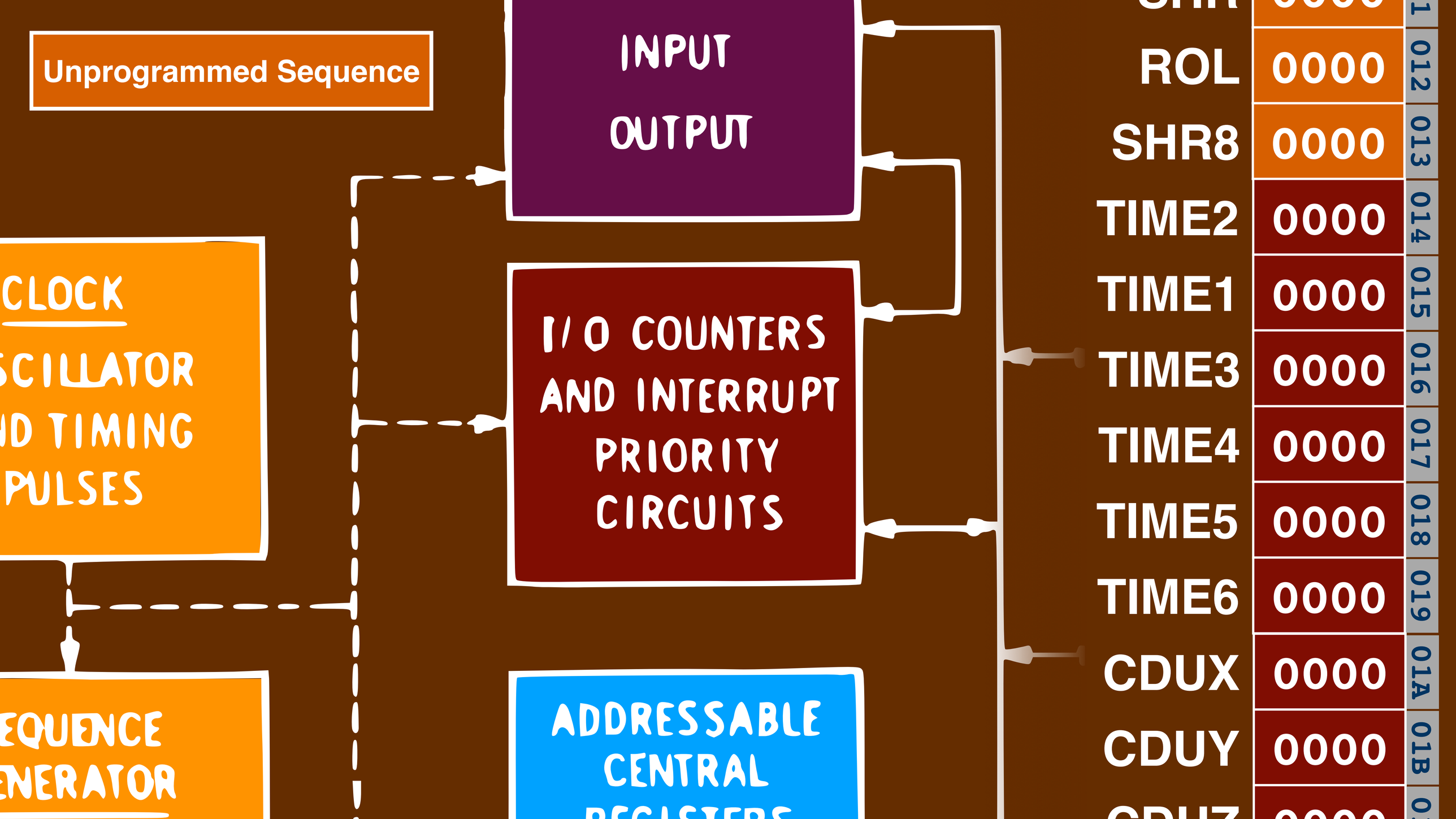
```
 4. xx  L2GD  RB    WYD   A2X   PIFL
 5. 0x  RG    WL    TSGU  DVST  CLXC
 5. 1x  RG    WL    TSGU  DVST  RB1F
 6. xx  RU    WB
 7. xx  L2GD  RB    WYD   A2X   PIFL
 8. 0x  RG    WL    TSGU  DVST  CLXC
 8. 1x  RG    WL    TSGU  DVST  RB1F
 9. xx  RU    WB
10. xx  L2GD  RB    WYD   A2X   PIFL
11. 0x  RG    WL    TSGU  DVST  CLXC
11. 1x  RG    WL    TSGU  DVST  RB1F
12. xx  RU    WB
 1. xx  L2GD  RB    WYD   A2X   PIFL
 2. 0x  RG    WL    TSGU  DVST  CLXC
 2. 1x  RG    WL    TSGU  DVST  RB1F
 3. xx  RU    WB    STAGE
```

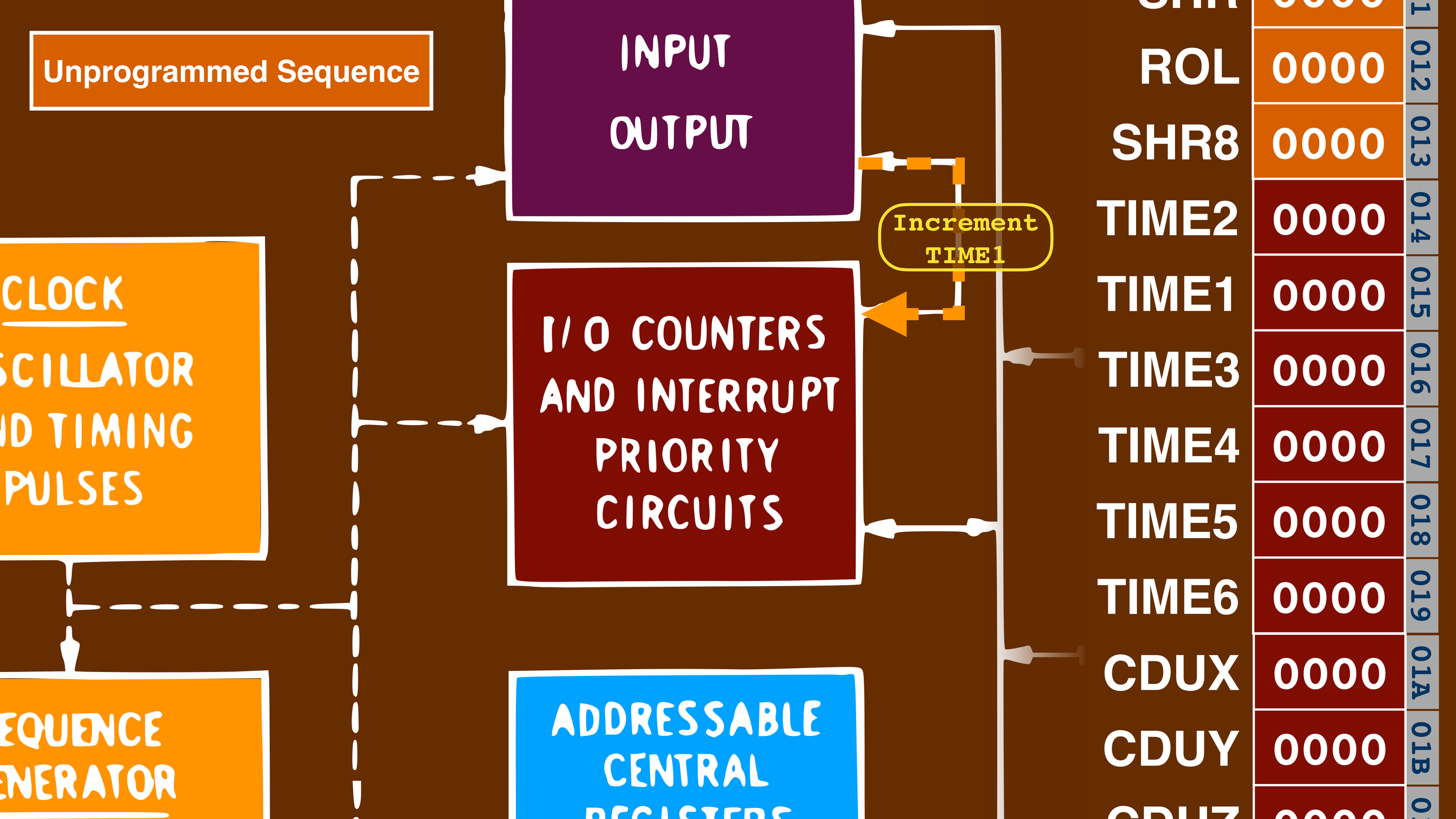
DV7

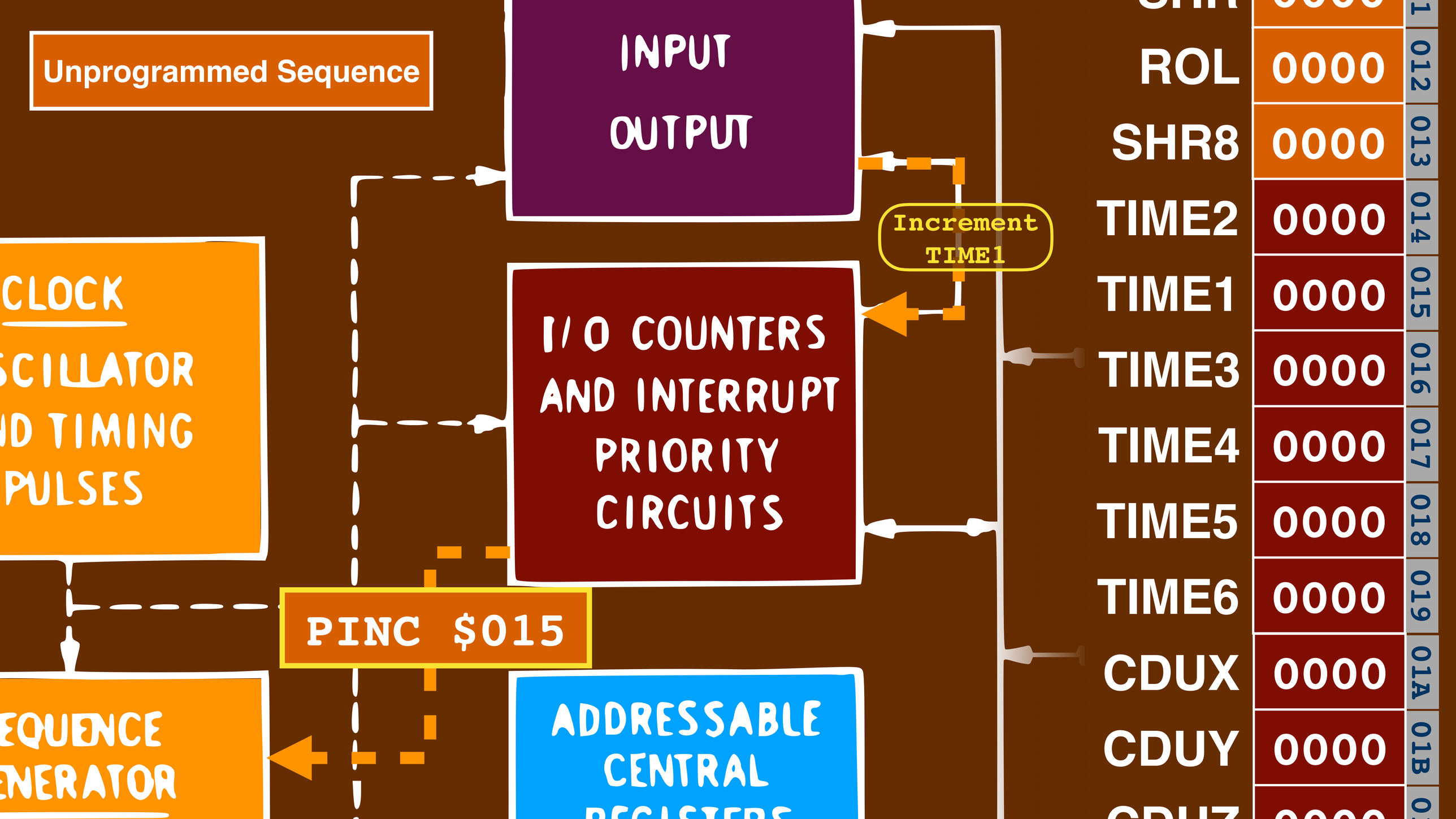
```
 4. xx  L2GD  RB    WYD   A2X   PIFL
 5. 0x  RG    WL    TSGU  DVST  CLXC
 5. 1x  RG    WL    TSGU  DVST  RB1F
 6. xx  RU    WB
 7. xx  L2GD  RB    WYD   A2X   PIFL
 8. 0x  RG    WL    TSGU  DVST  CLXC
 8. 1x  RG    WL    TSGU  DVST  RB1F
 9. xx  RU    WB
10. xx  L2GD  RB    WYD   A2X   PIFL
11. 0x  RG    WL    TSGU  DVST  CLXC
11. 1x  RG    WL    TSGU  DVST  RB1F
12. xx  RU    WB
 1. xx  L2GD  RB    WYD   A2X   PIFL
 2. 0x  RG    WL    TSGU  DVST  CLXC
 2. 1x  RG    WL    TSGU  DVST  RB1F
 3. xx  RU    WB    STAGE
```

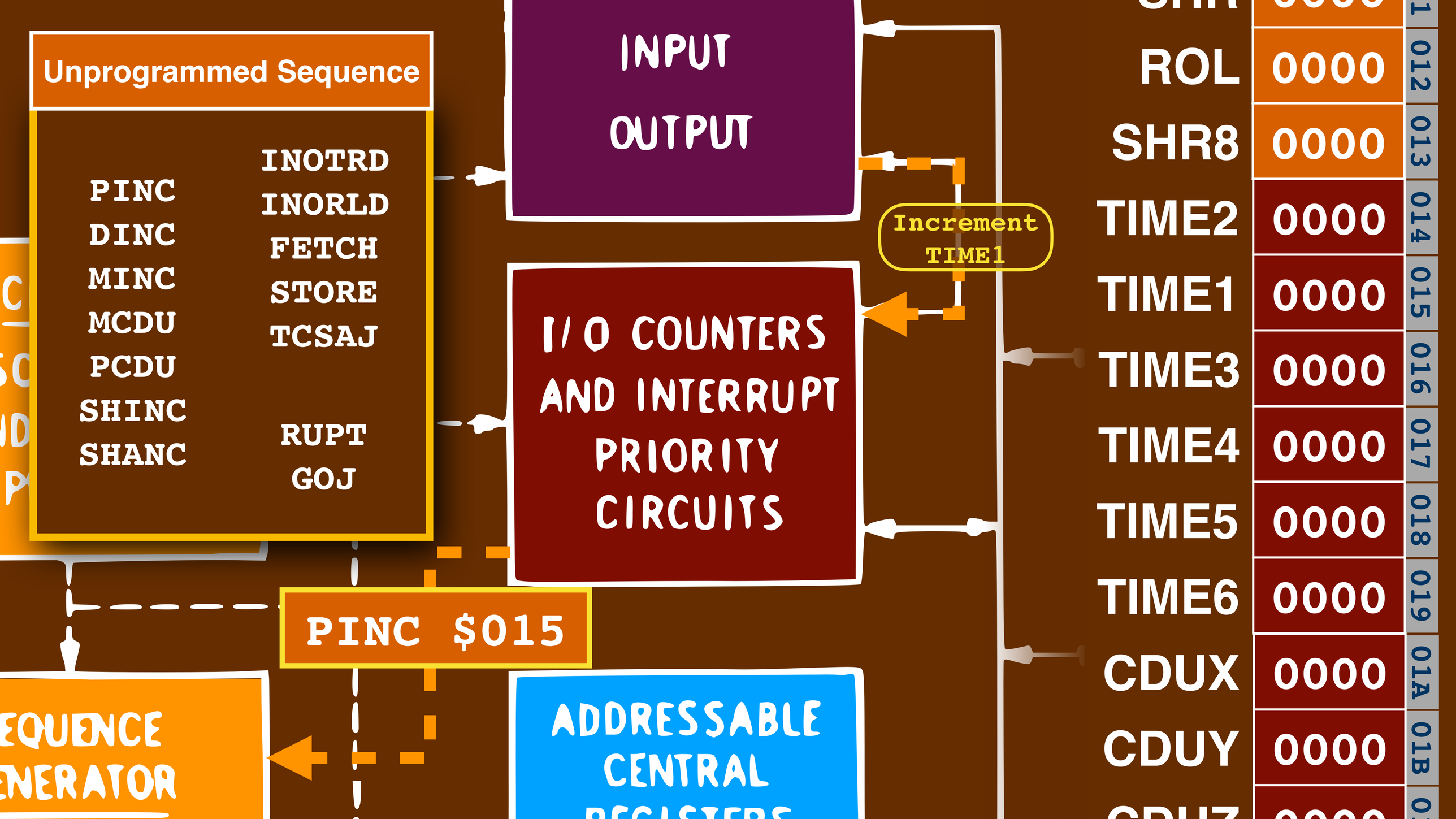

Unprogrammed Sequence
DATA PATHS
CONTROL PATHS
INPUT
OUTPUT
CLOCK
OSCILLATOR AND TIMING PULSES
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
FIXED MEMORY
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
G
MEMORY LOCAL REGISTER
ERASABLE MEMORY
PARITY
MEMORY TIMING
ARITHMETIC UNIT
WRITE BUS
SQ
INSTRUCTION DECODER
B
BUFFER

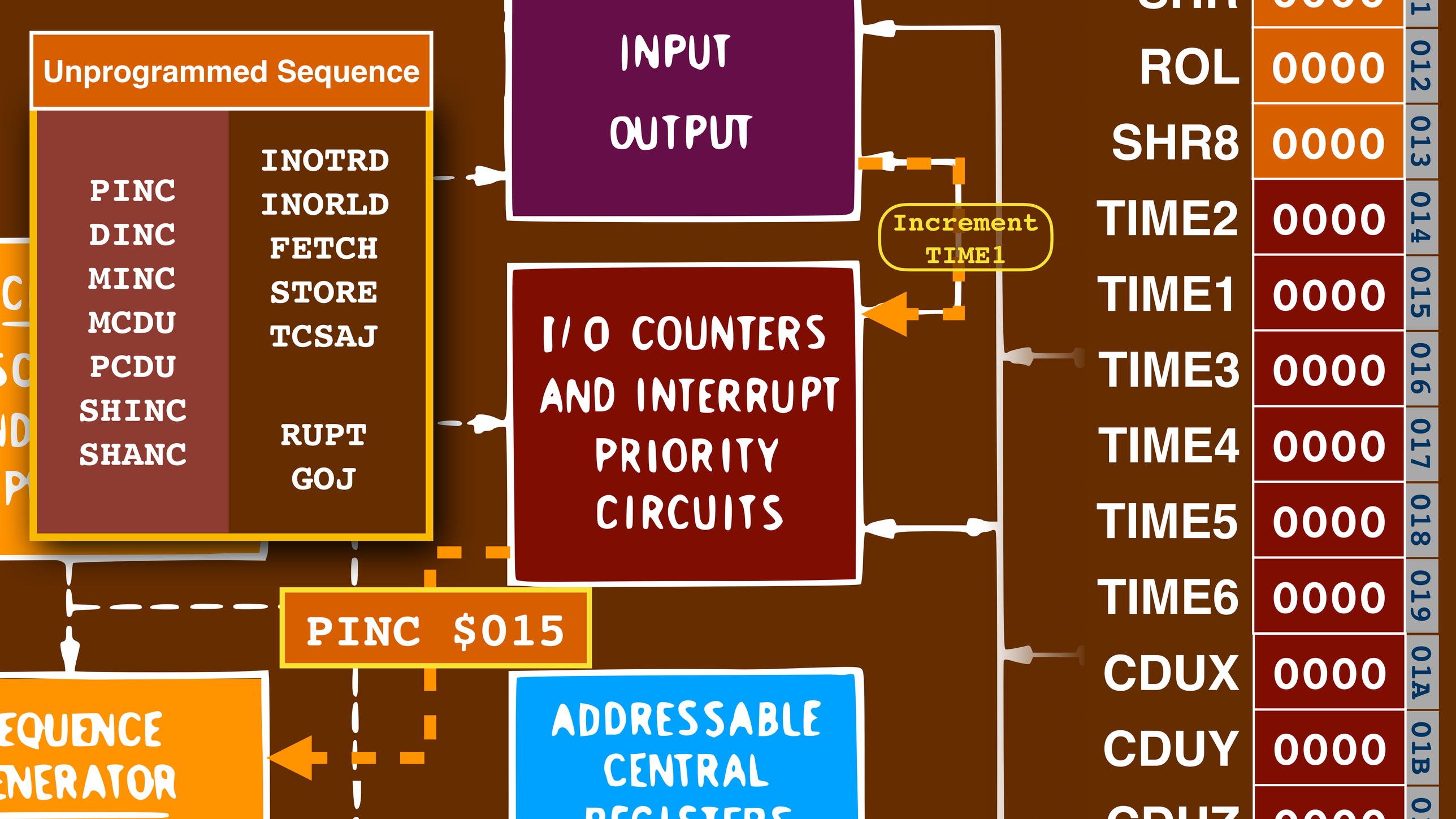

Unprogrammed Sequence
INPUT
OUTPUT
CLOCK
SCILLATOR
ND TIMING
PULSES
I/O COUNTERS
AND INTERRUPT
PRIORITY
CIRCUITS
MEMORY
BANK
REGISTERS
S
MEMORY
ADDRESS
REGISTER
EQUENCE
ENERATOR
ADDRESSABLE
CENTRAL

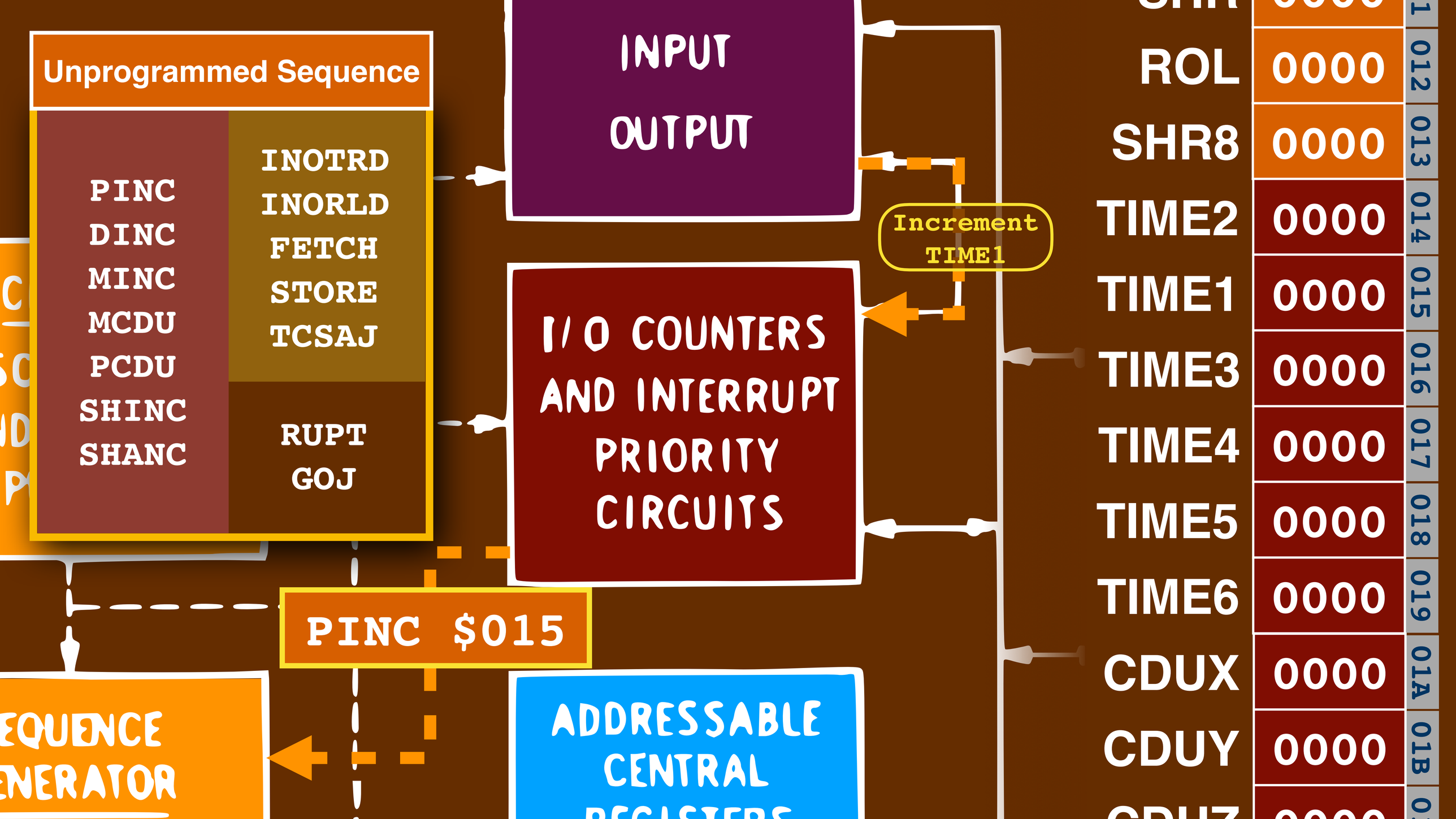

Unprogrammed Sequence
INPUT
OUTPUT
CLOCK
SCILLATOR
ND TIMING
PULSES
I/O COUNTERS
AND INTERRUPT
PRIORITY
CIRCUITS
EQUENCE
ENERATOR
ADDRESSABLE
CENTRAL
SHR
ROL 0000
SHR8 0000
TIME2 0000
TIME1 0000
TIME3 0000
TIME4 0000
TIME5 0000
TIME6 0000
CDUX 0000
CDUY 0000
012
013
014
015
016
017
018
019
01A
01B

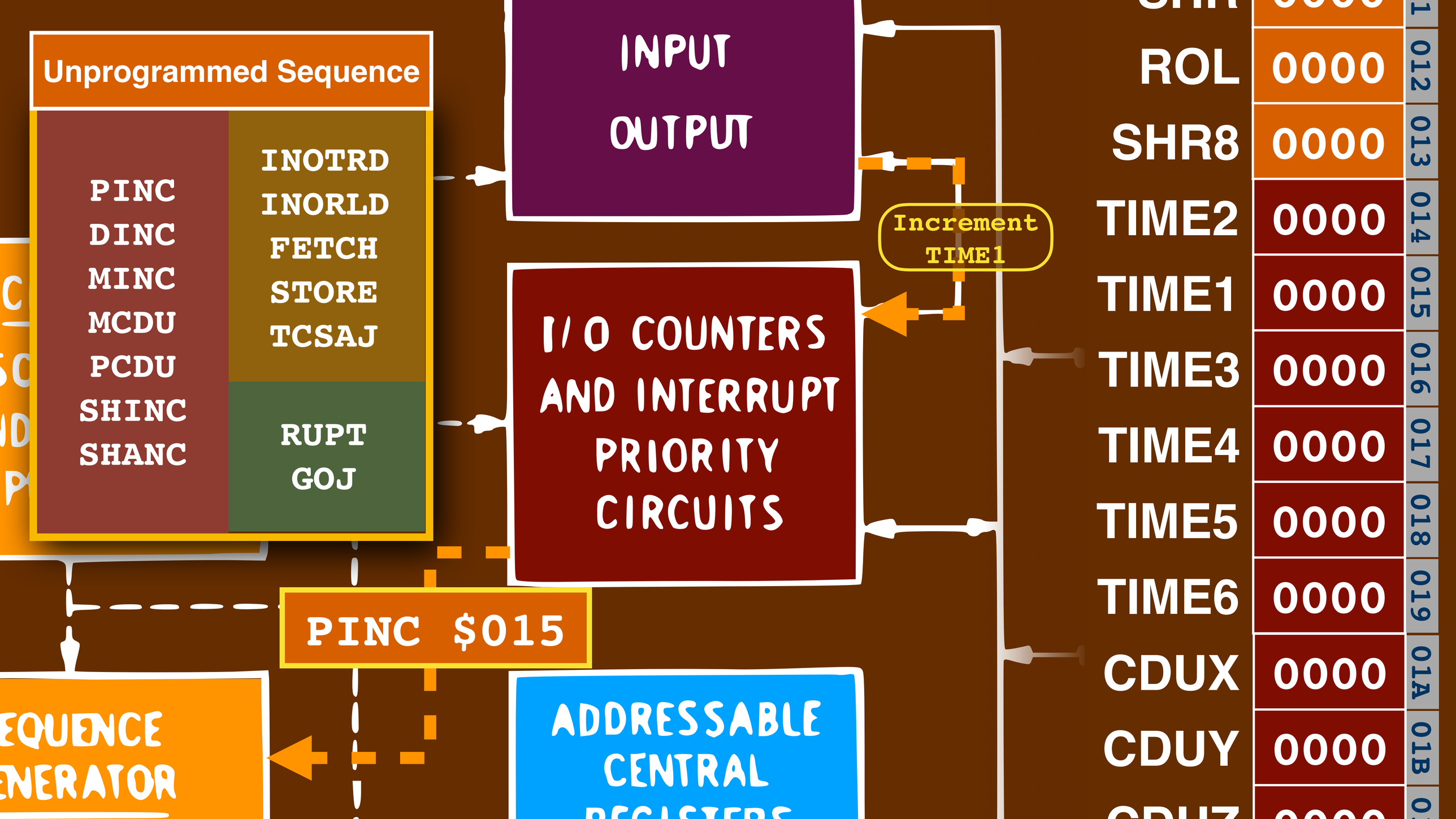

Unprogrammed Sequence
INPUT
OUTPUT
Increment TIME1
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
CLOCK
SCILLATOR
ND TIMING
PULSES
EQUENCE
ENERATOR
ADDRESSABLE CENTRAL
ROL 0000 012
SHR8 0000 013
TIME2 0000 014
TIME1 0000 015
TIME3 0000 016
TIME4 0000 017
TIME5 0000 018
TIME6 0000 019
CDUX 0000 01A
CDUY 0000 01B


Unprogrammed Sequence
INPUT
OUTPUT
Increment TIME1
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
CLOCK
SCILLATOR
ND TIMING
PULSES
PINC $015
EQUENCE
ENERATOR
ADDRESSABLE CENTRAL
SHR
ROL 0000
SHR8 0000
TIME2 0000
TIME1 0000
TIME3 0000
TIME4 0000
TIME5 0000
TIME6 0000
CDUX 0000
CDUY 0000
011
012
013
014
015
016
017
018
019
01A
01B

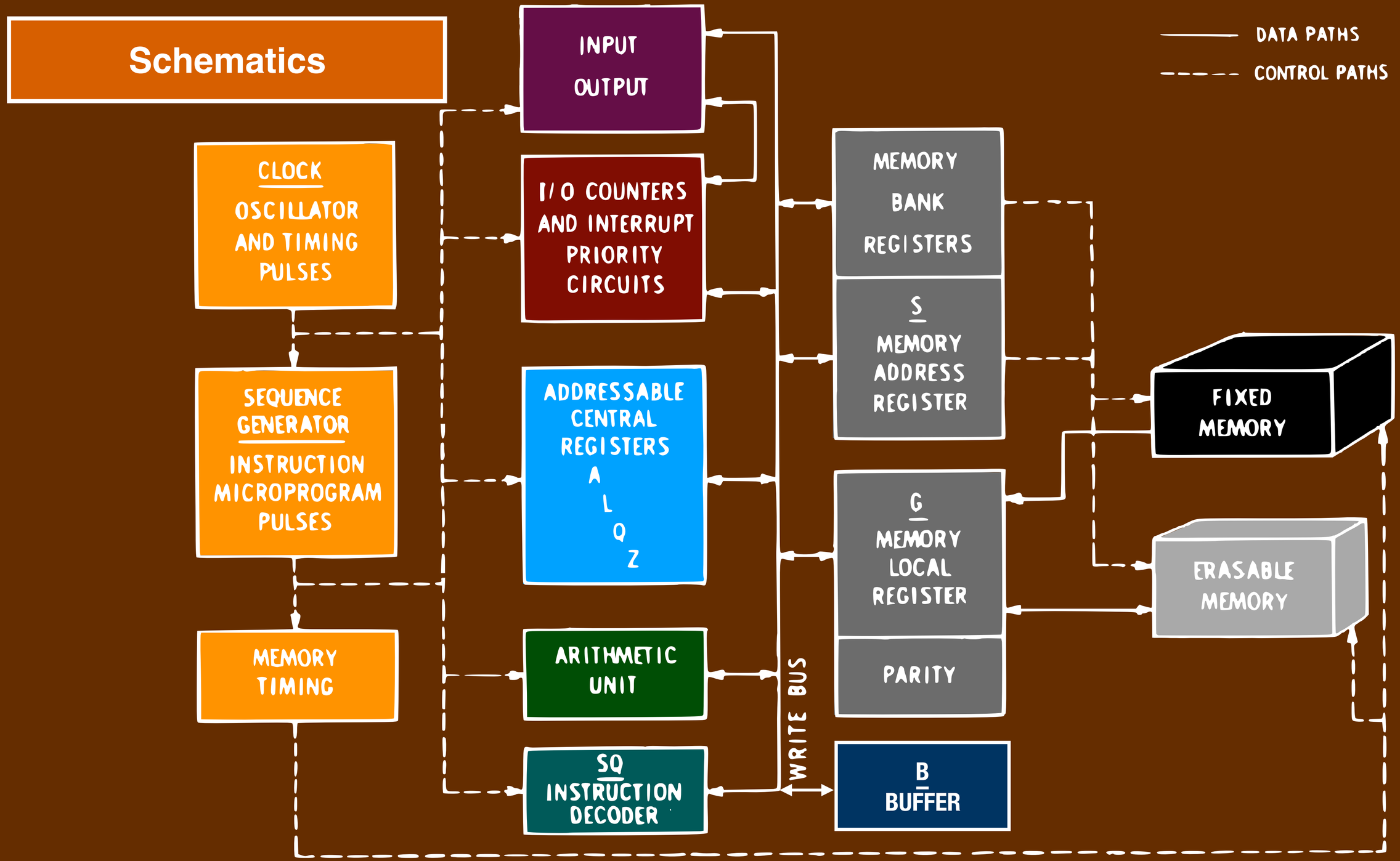

Unprogrammed Sequence
PINC
DINC
MINC
MCDU
PCDU
SHINC
SHANC
INOTRD
INORLD
FETCH
STORE
TCSAJ
RUPT
GOJ
INPUT
OUTPUT
Increment TIME1
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
PINC $015
ADDRESSABLE CENTRAL
EQUENCE
ENERATOR
SHR
ROL 0000
SHR8 0000
TIME2 0000
TIME1 0000
TIME3 0000
TIME4 0000
TIME5 0000
TIME6 0000
CDUX 0000
CDUY 0000
012
013
014
015
016
017
018
019
01A
01B


INPUT
OUTPUT
Unprogrammed Sequence
PINC
DINC
MINC
MCDU
PCDU
SHINC
SHANC
INOTRD
INORLD
FETCH
STORE
TCSAJ
RUPT
GOJ
Increment TIME1
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
PINC $015
EQUENCE
ENERATOR
ADDRESSABLE CENTRAL
ROL 0000
SHR8 0000
TIME2 0000
TIME1 0000
TIME3 0000
TIME4 0000
TIME5 0000
TIME6 0000
CDUX 0000
CDUY 0000
012
013
014
015
016
017
018
019
01A
01B

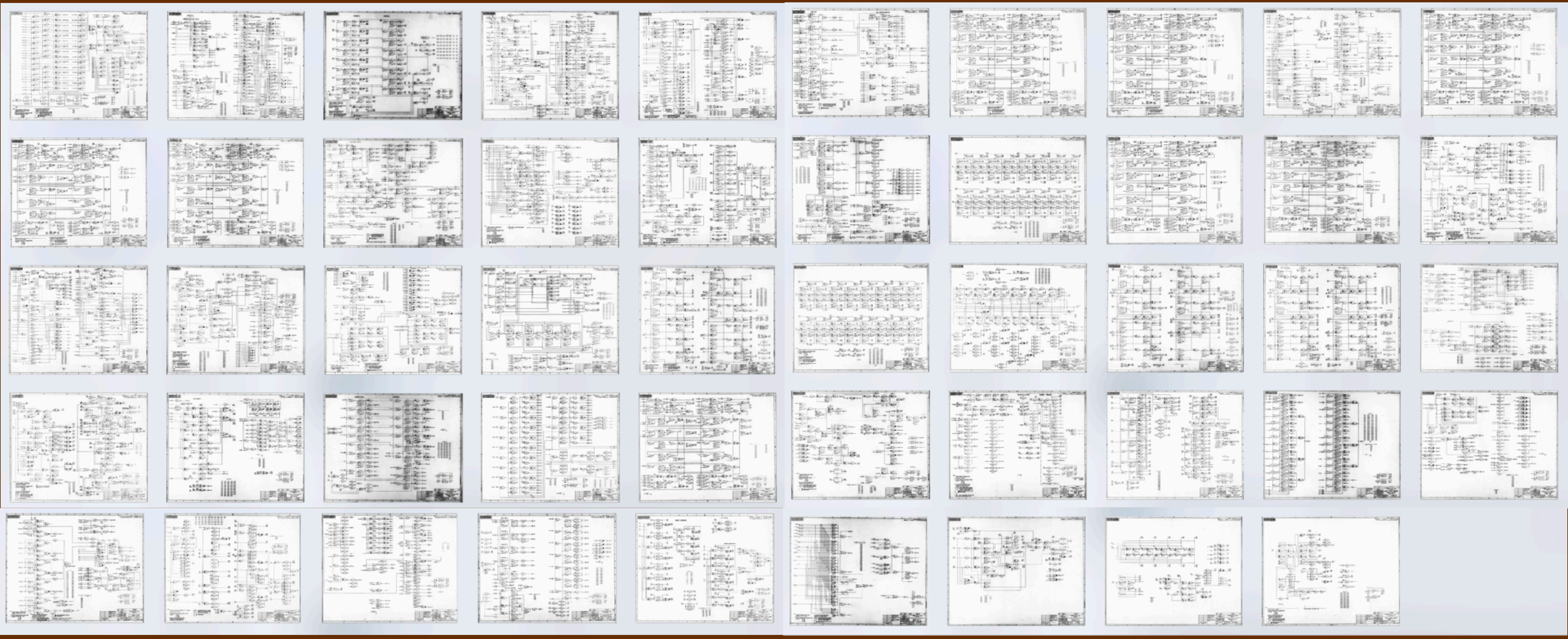

Unprogrammed Sequence
PINC
DINC
MINC
MCDU
PCDU
SHINC
SHANC
INOTRD
INORLD
FETCH
STORE
TCSAJ
RUPT
GOJ
INPUT
OUTPUT
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
Increment TIME1
PINC $015
ADDRESSABLE CENTRAL
EQUENCE
ENERATOR
SHR
ROL 0000
SHR8 0000
TIME2 0000
TIME1 0000
TIME3 0000
TIME4 0000
TIME5 0000
TIME6 0000
CDUX 0000
CDUY 0000
012
013
014
015
016
017
018
019
01A
01B


Unprogrammed Sequence
PINC
DINC
MINC
MCDU
PCDU
SHINC
SHANC
INOTRD
INORLD
FETCH
STORE
TCSAJ
RUPT
GOJ
INPUT
OUTPUT
Increment TIME1
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
PINC $015
ADDRESSABLE CENTRAL
EQUENCE
ENERATOR
ROL 0000
SHR8 0000
TIME2 0000
TIME1 0000
TIME3 0000
TIME4 0000
TIME5 0000
TIME6 0000
CDUX 0000
CDUY 0000
012
013
014
015
016
017
018
019
01A
01B

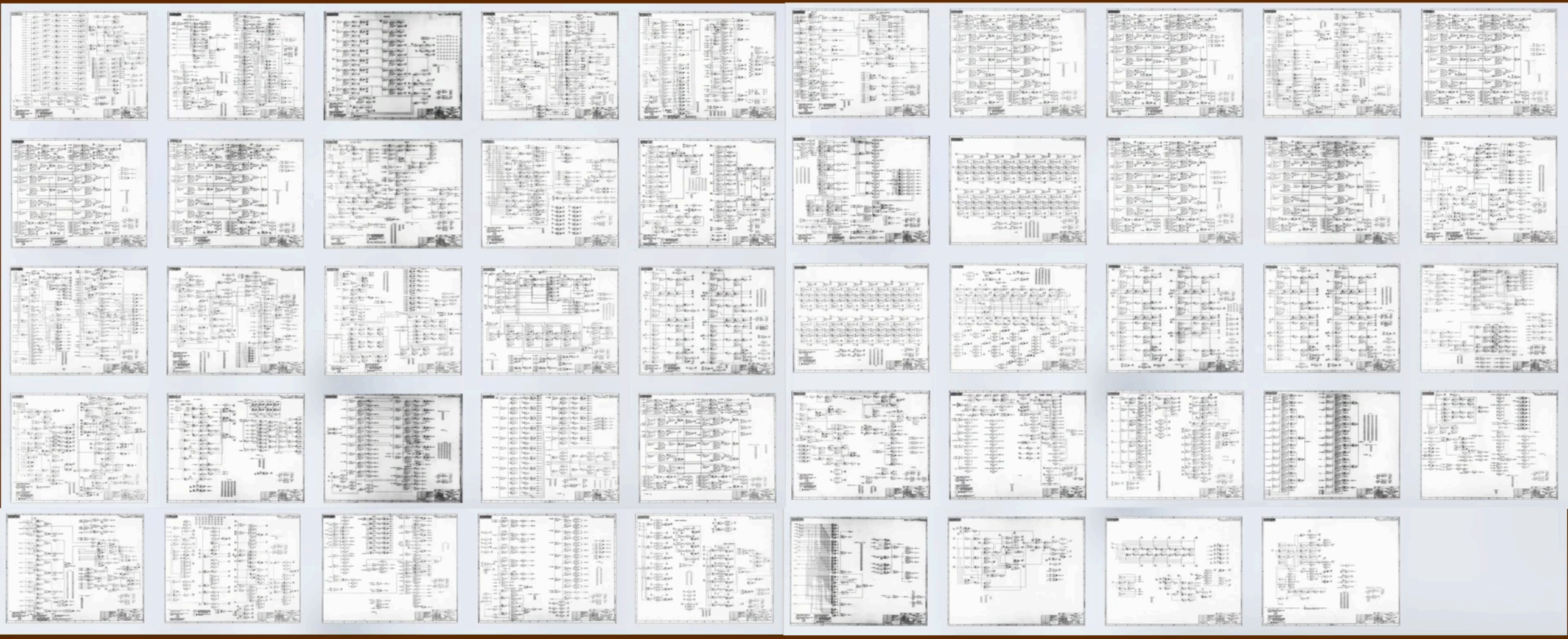
Schematics


DATA PATHS
CONTROL PATHS
INPUT
OUTPUT
CLOCK
OSCILLATOR AND TIMING PULSES
I/O COUNTERS AND INTERRUPT PRIORITY CIRCUITS
MEMORY BANK REGISTERS
S
MEMORY ADDRESS REGISTER
SEQUENCE GENERATOR
INSTRUCTION MICROPROGRAM PULSES
ADDRESSABLE CENTRAL REGISTERS
A
L
Q
Z
FIXED MEMORY
G
MEMORY LOCAL REGISTER
ERASABLE MEMORY
MEMORY TIMING
ARITHMETIC UNIT
PARITY
WRITE BUS
SQ
INSTRUCTION DECODER
B
BUFFER

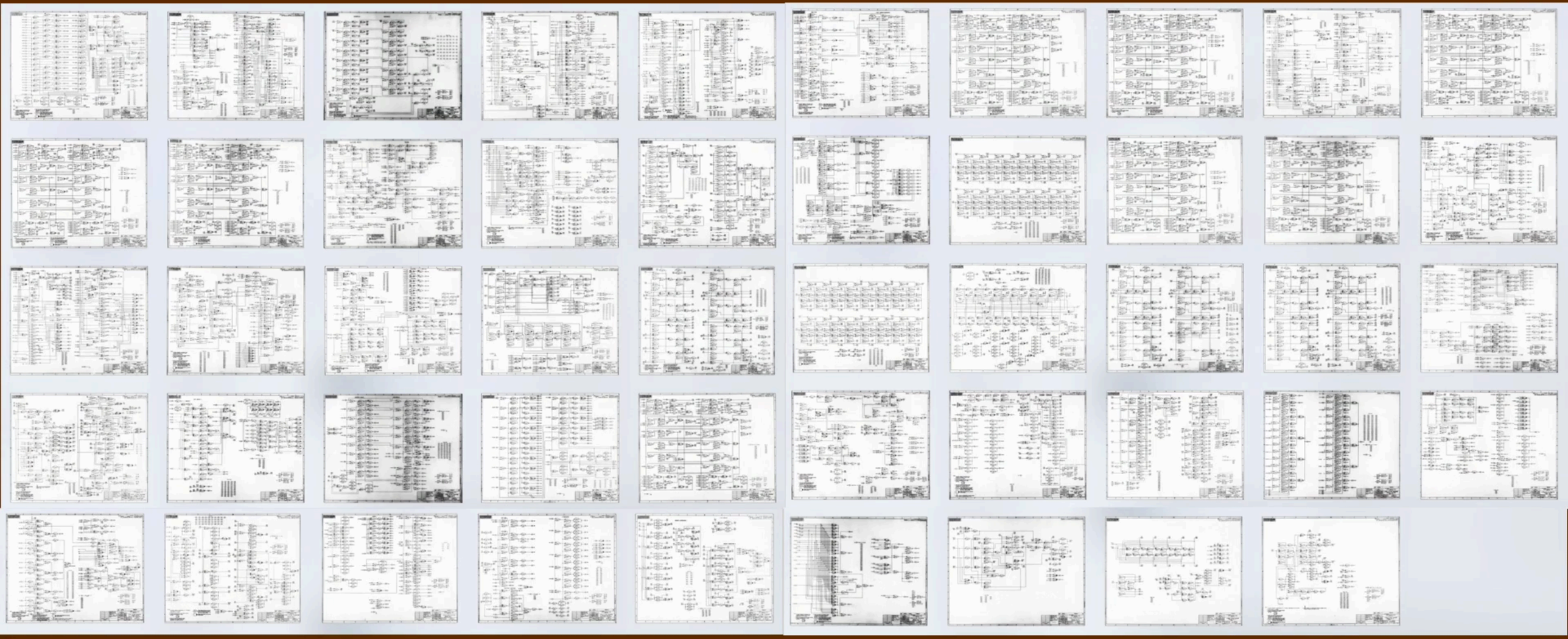
Schematics



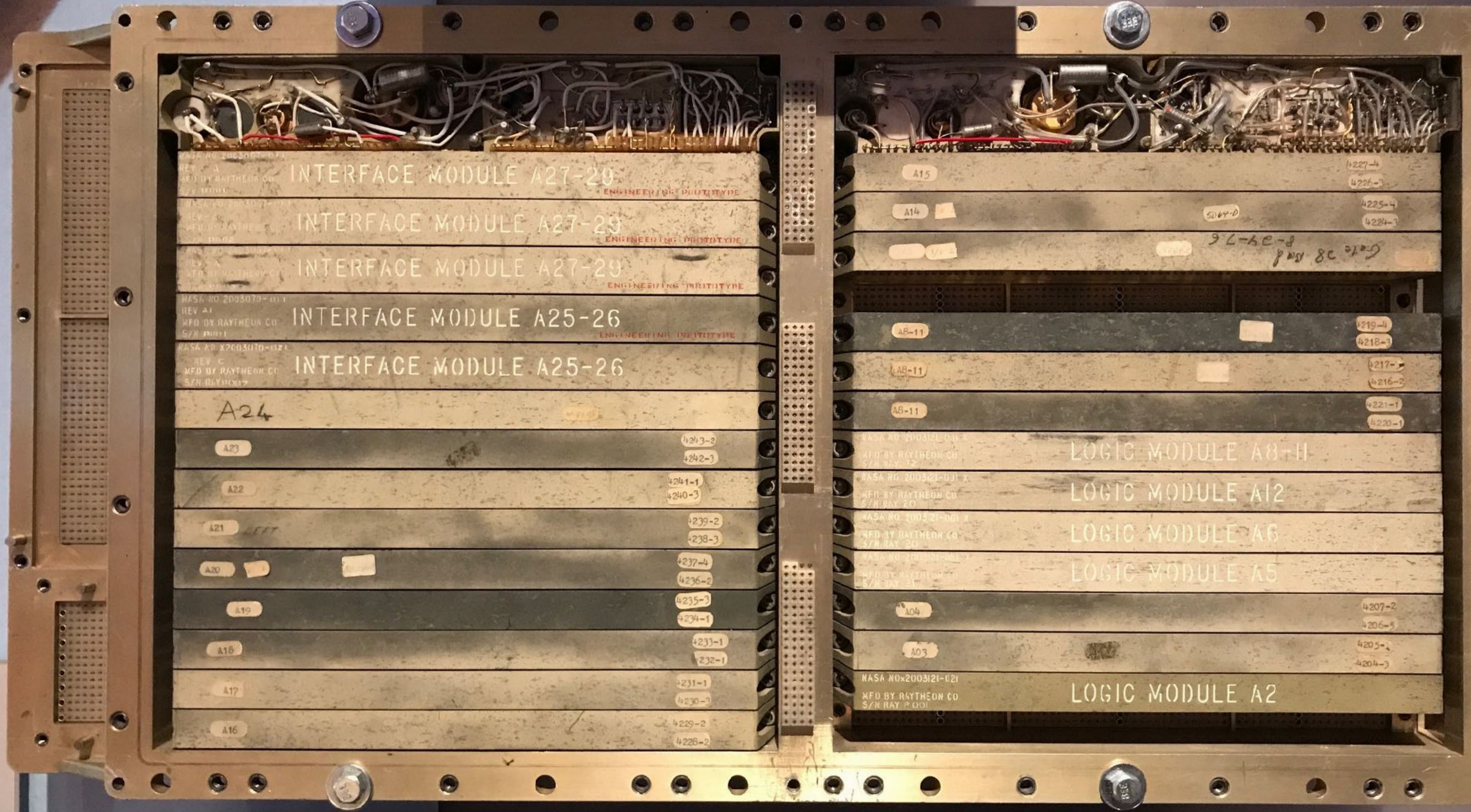
Schematics



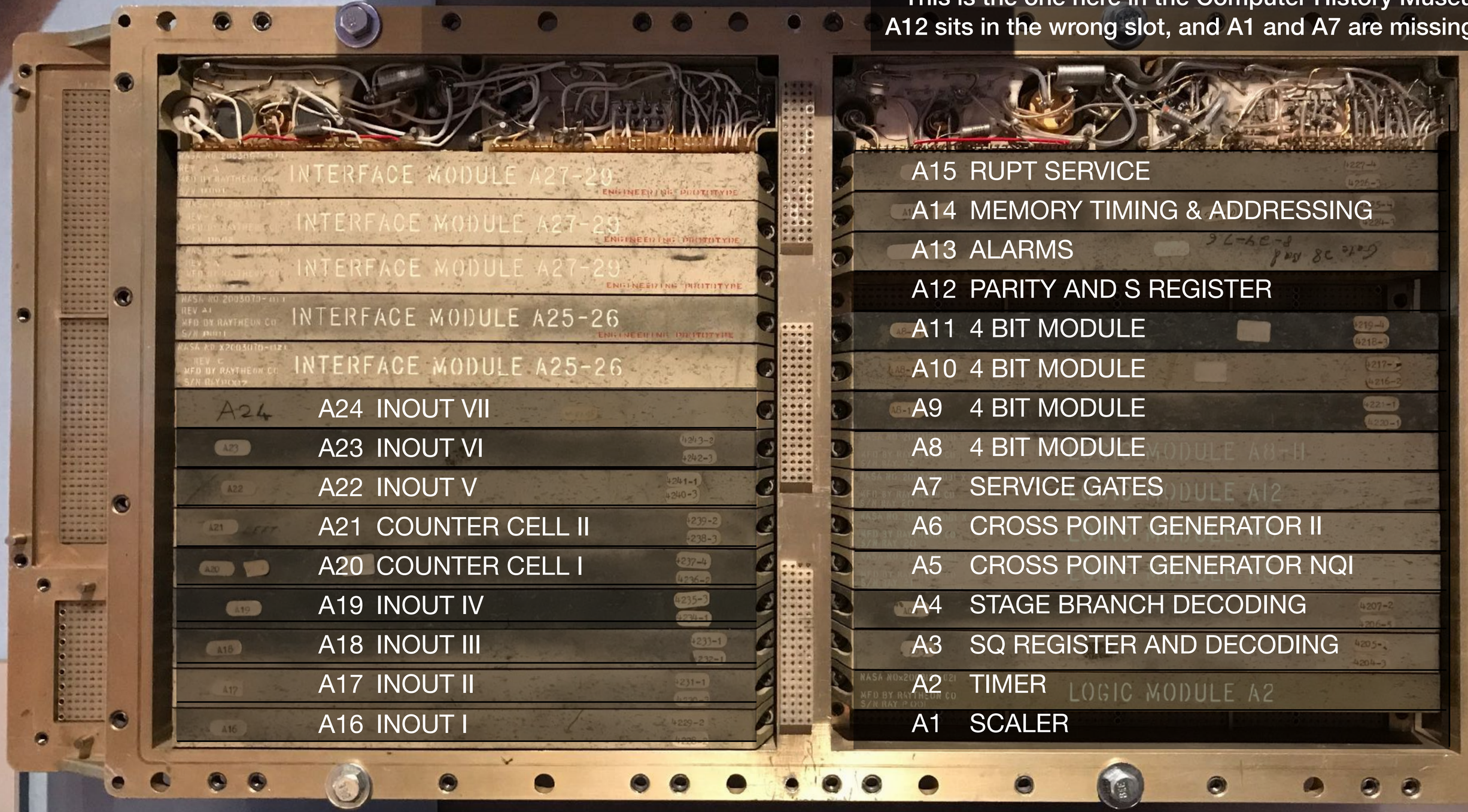
Schematics

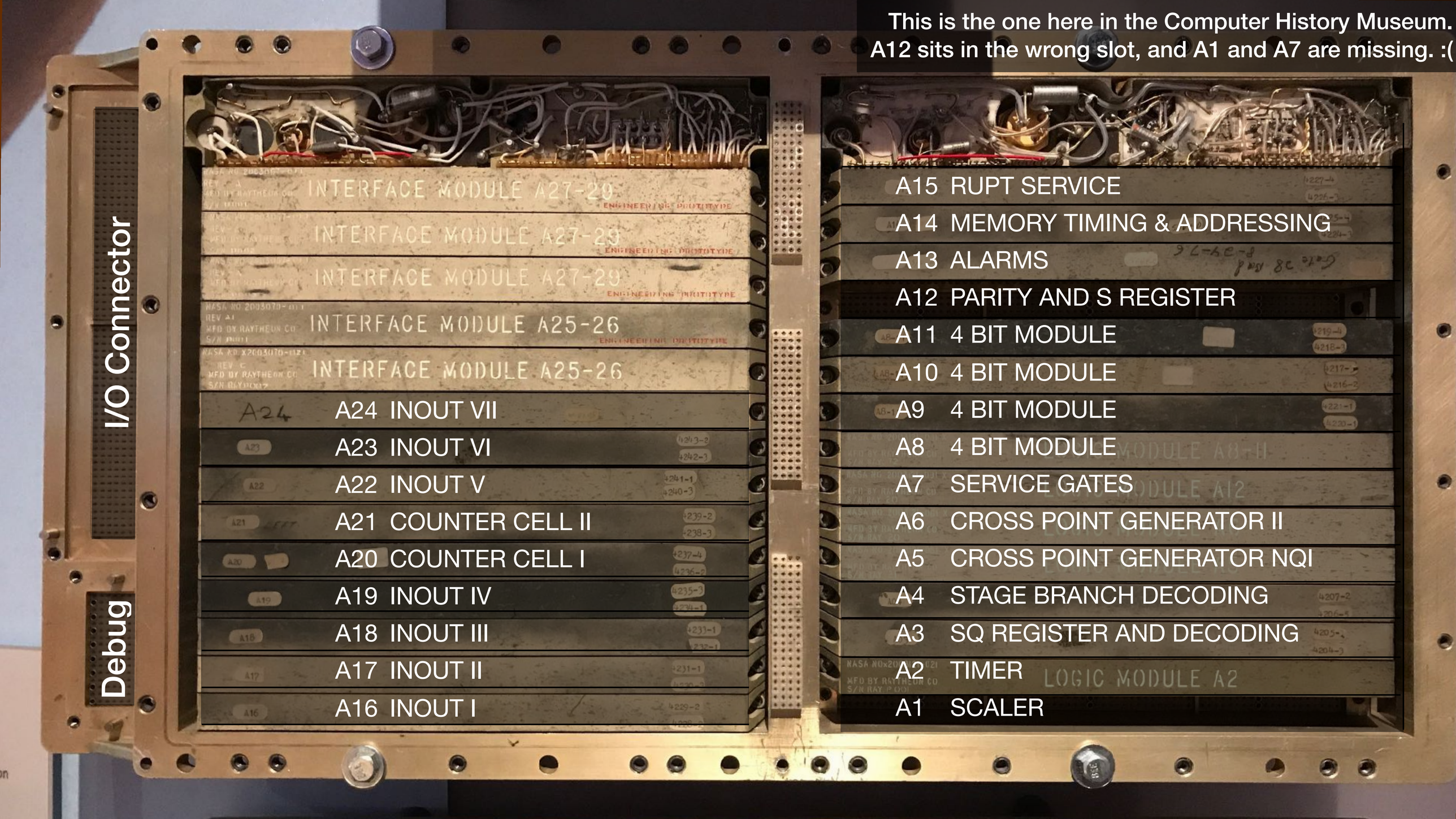



A8-11
A8-11
A8-11
4219-4
4218-3
4217-3
4216-2
4221-1
4220-1
NASA NO 2003121-081 X
MFD BY RAYTHEON CO
S/N RAY 72
LOGIC MODULE A8-11
NASA NO 2003121-091 X
MFD BY RAYTHEON CO
S/N RAY 20
LOGIC MODULE A12
NASA NO 2003121-061 X
MFD BY RAYTHEON CO
S/N RAY 20
LOGIC MODULE A6
NASA NO 2003121-051 X
MFD BY RAYTHEON CO
S/N RAY 21
LOGIC MODULE A5
A04
4207-2
4206-5
A03
4205-2
4204-3
NASA NO×2003121-021
MFD BY RAYTHEON CO
S/N RAY. P 001
LOGIC MODULE A2


INTERFACE MODULE A27-29
ENGINEERING PROTOTYPE
INTERFACE MODULE A27-29
ENGINEERING PROTOTYPE
INTERFACE MODULE A27-29
ENGINEERING PROTOTYPE
INTERFACE MODULE A25-26
INTERFACE MODULE A25-26
A24
A23
A22
A21
A20
A19
A18
A17
A16
A15
A14
A8-11
A8-11
A8-11
LOGIC MODULE A8-11
LOGIC MODULE A12
LOGIC MODULE A6
LOGIC MODULE A5
A04
A03
LOGIC MODULE A2

This is the one here in the Computer History Museum.
A12 sits in the wrong slot, and A1 and A7 are missing. :(




This is the one here in the Computer History Museum.
A12 sits in the wrong slot, and A1 and A7 are missing. :(
I/O Connector
Debug
INTERFACE MODULE A27-29
INTERFACE MODULE A27-29
INTERFACE MODULE A27-29
INTERFACE MODULE A25-26
INTERFACE MODULE A25-26
A24 INOUT VII
A23 INOUT VI
A22 INOUT V
A21 COUNTER CELL II
A20 COUNTER CELL I
A19 INOUT IV
A18 INOUT III
A17 INOUT II
A16 INOUT I
A15 RUPT SERVICE
A14 MEMORY TIMING & ADDRESSING
A13 ALARMS
A12 PARITY AND S REGISTER
A11 4 BIT MODULE
A10 4 BIT MODULE
A9 4 BIT MODULE
A8 4 BIT MODULE
A7 SERVICE GATES
A6 CROSS POINT GENERATOR II
A5 CROSS POINT GENERATOR NQI
A4 STAGE BRANCH DECODING
A3 SQ REGISTER AND DECODING
A2 TIMER
A1 SCALER

ALARM MODULE B8

ERASABLE DRIVER MODULE B9-10

ENGINEERING PROTOTYPE

ERASABLE DRIVER MODULE B9-10

CURRENT SWITCH MODULE B11

ENGINEERING PROTOTYPE

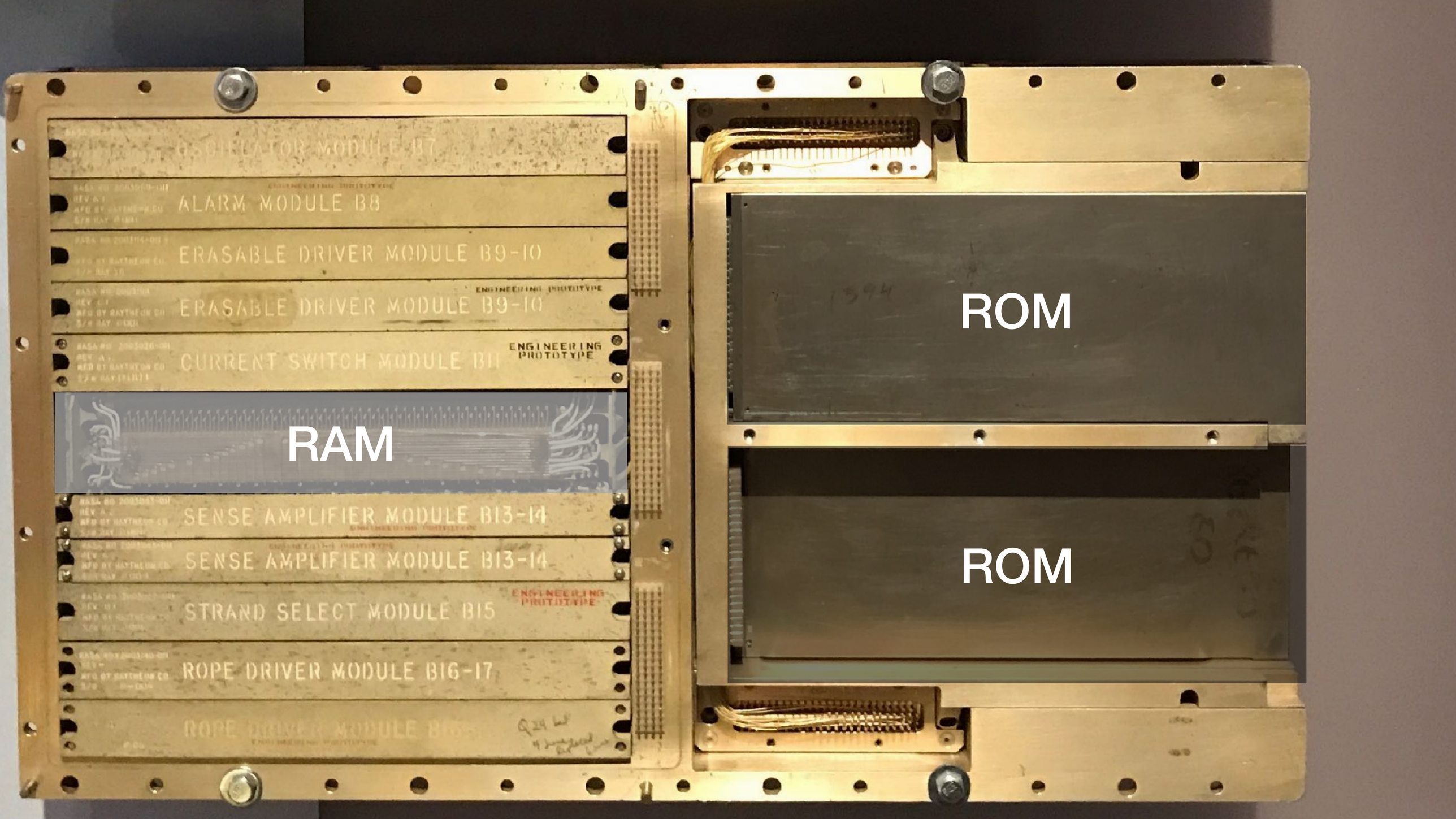
SENSE AMPLIFIER MODULE B13-14

SENSE AMPLIFIER MODULE B13-14

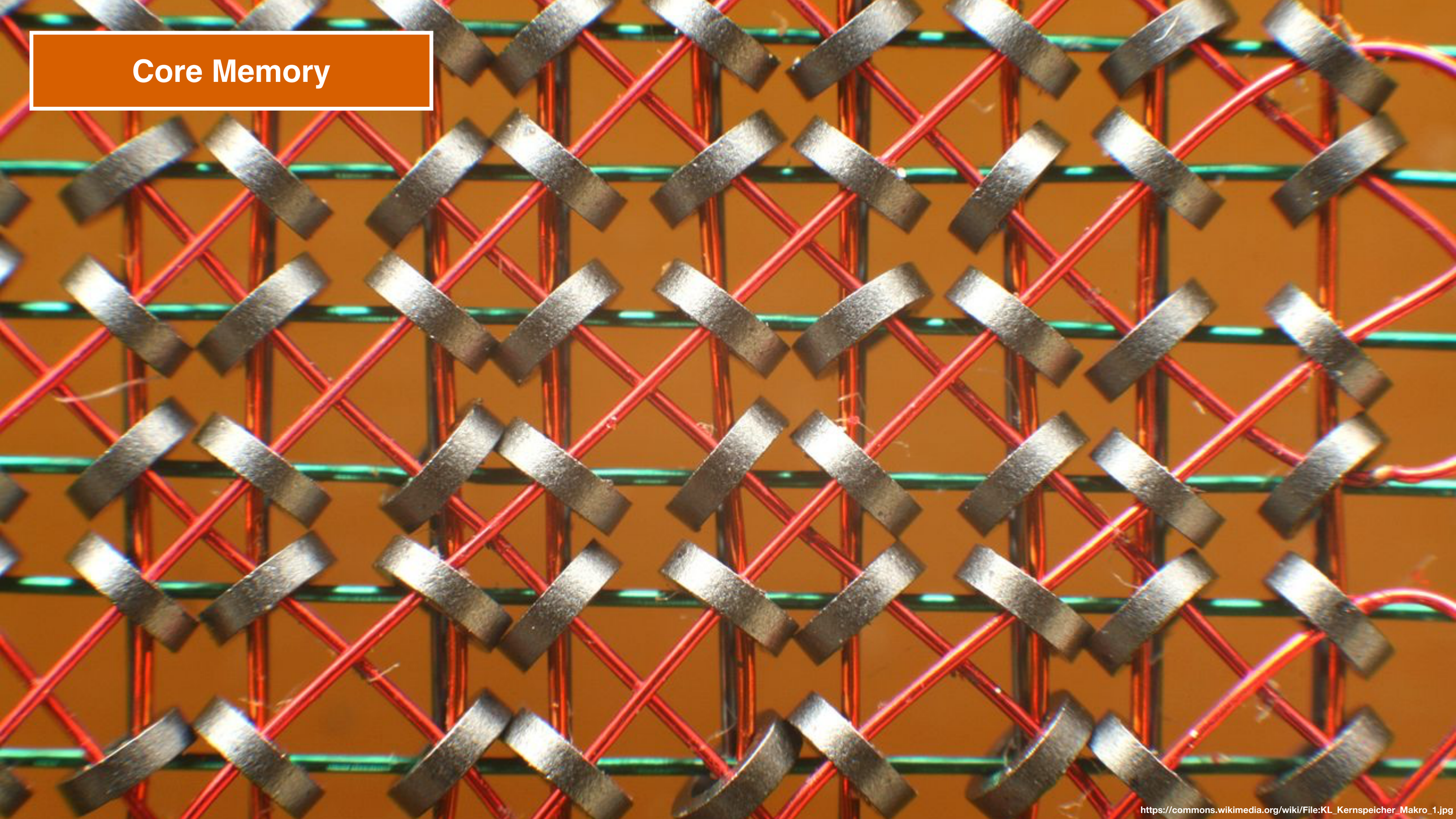
STRAND SELECT MODULE B15

ENGINEERING PROTOTYPE

ROPE DRIVER MODULE B16-17

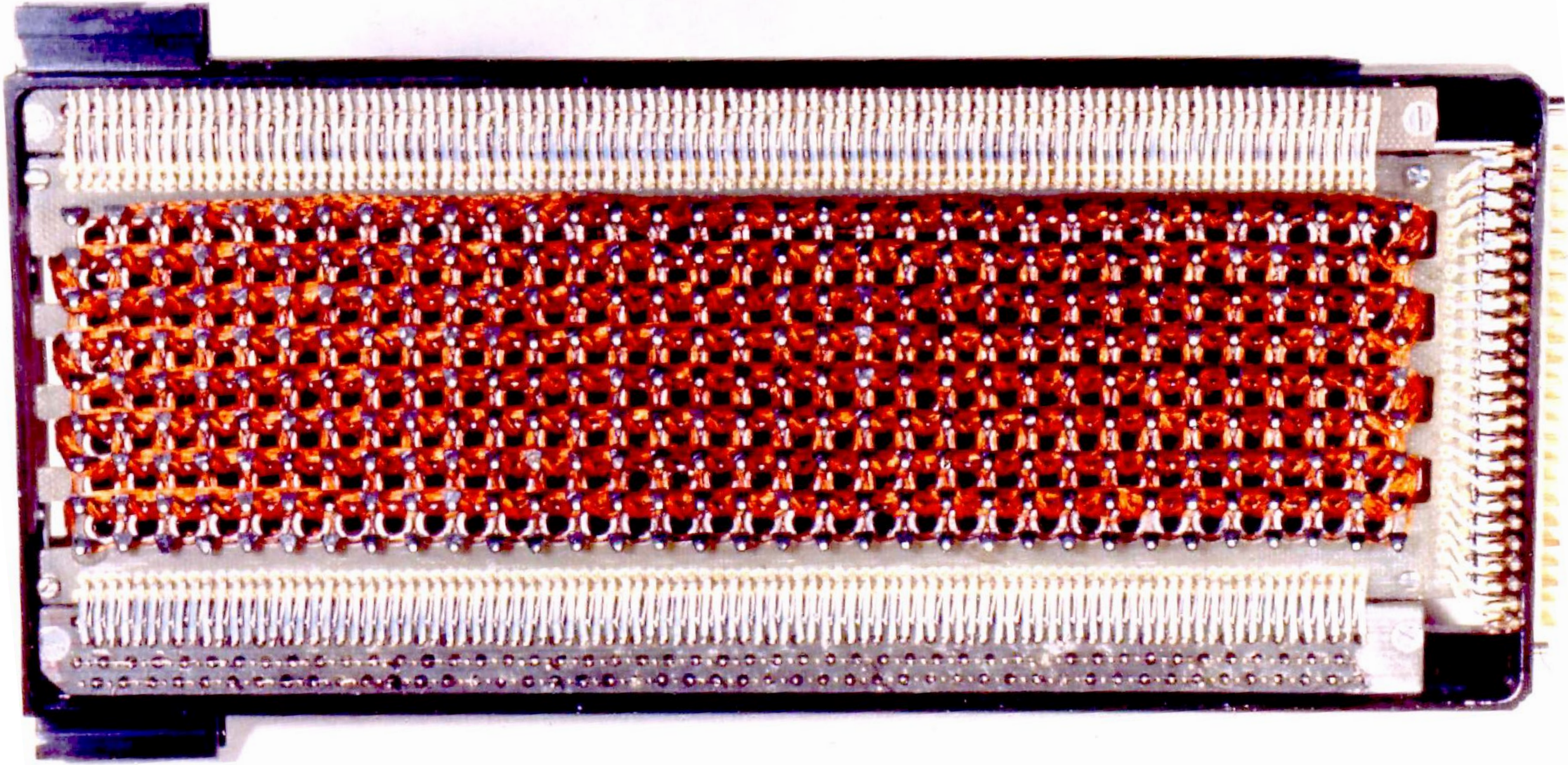

ALARM MODULE B8
ERASABLE DRIVER MODULE B9-10
ENGINEERING PROTOTYPE
ERASABLE DRIVER MODULE B9-10
ENGINEERING PROTOTYPE
CURRENT SWITCH MODULE B11
RAM
SENSE AMPLIFIER MODULE B13-14
SENSE AMPLIFIER MODULE B13-14
ENGINEERING PROTOTYPE
STRAND SELECT MODULE B15
ROPE DRIVER MODULE B16-17
ROM
ROM

Core Memory

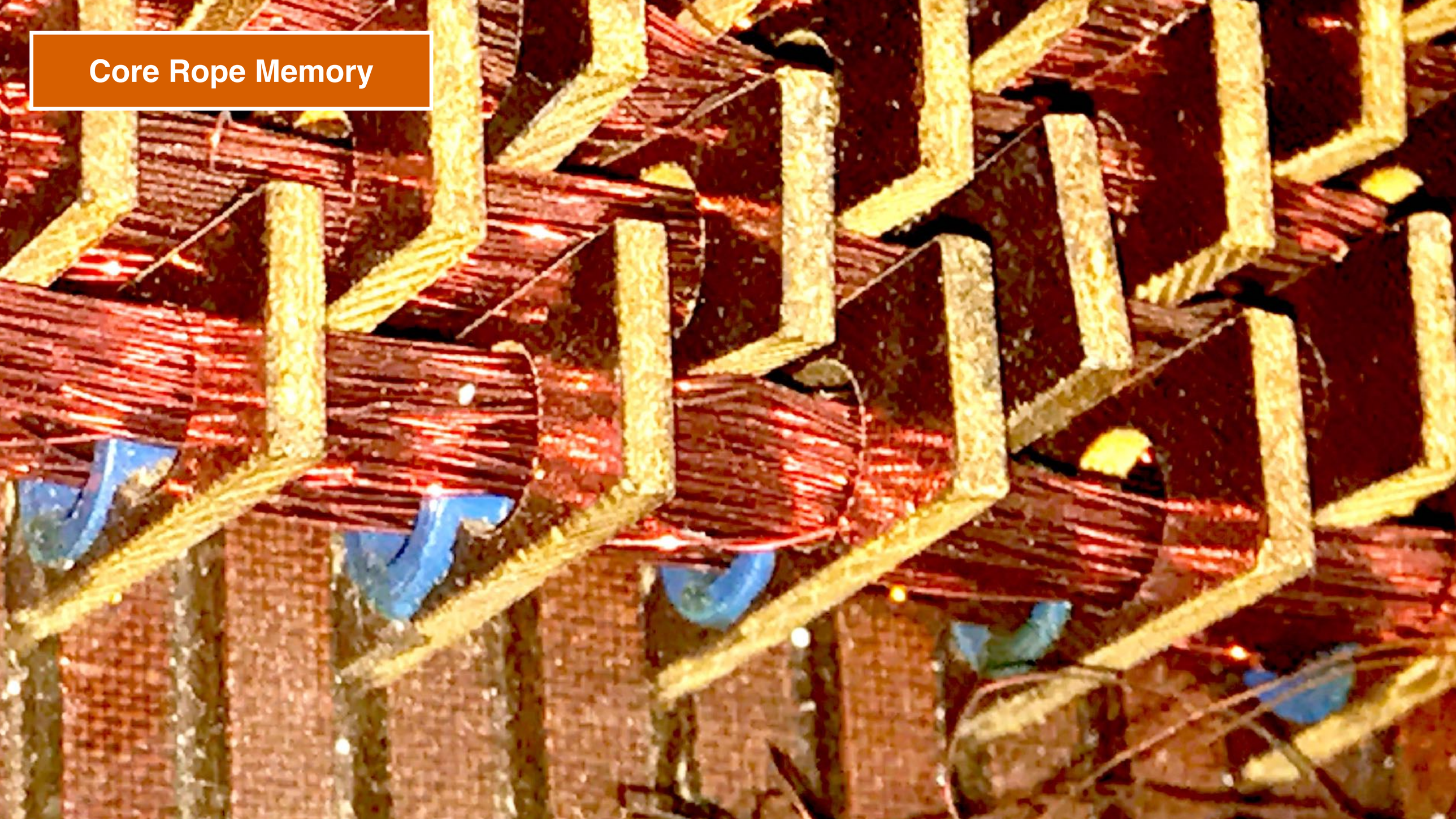


https://commons.wikimedia.org/wiki/File:KL_Kernspeicher_Makro_1.jpg

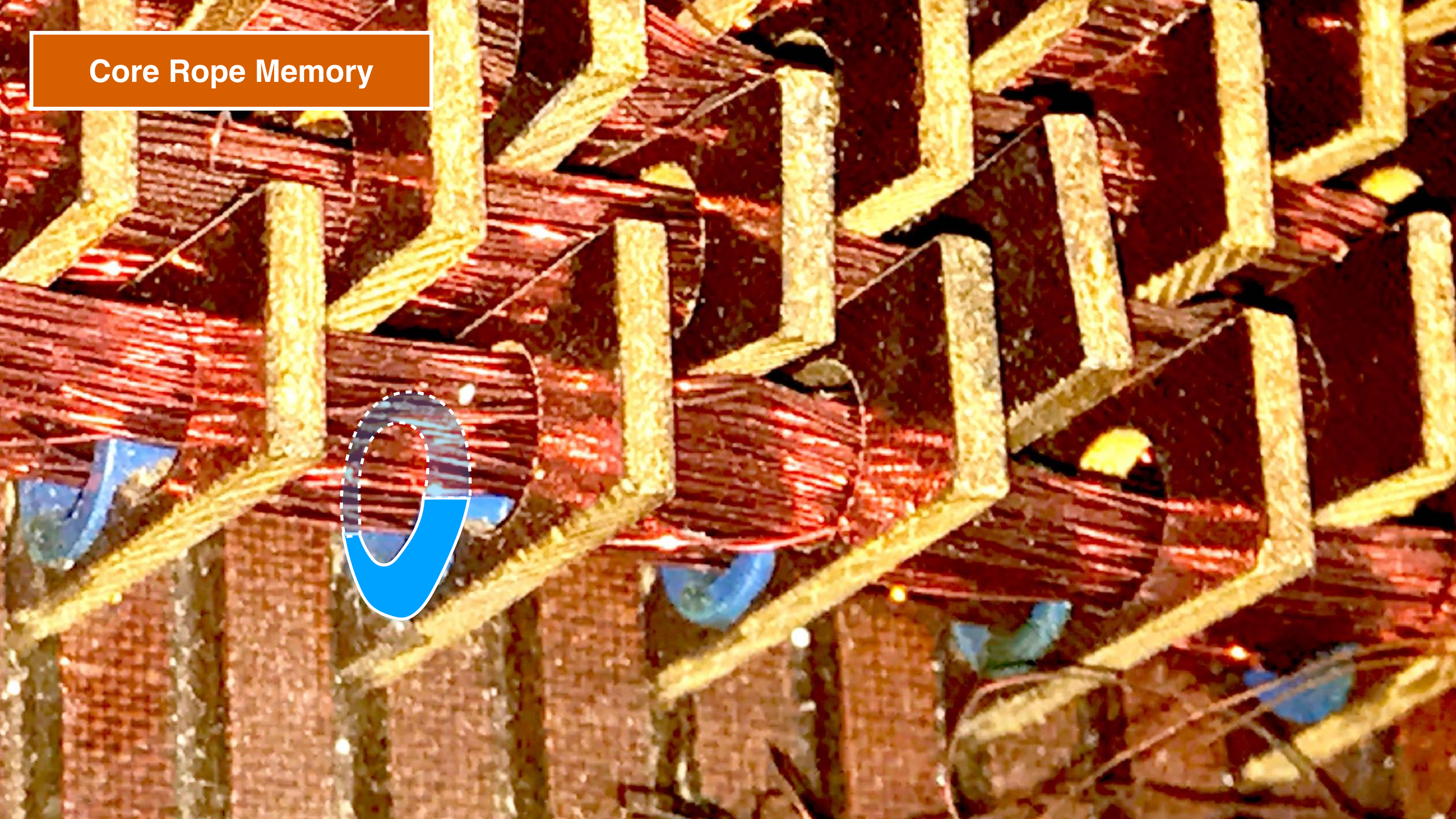
Core Rope Memory



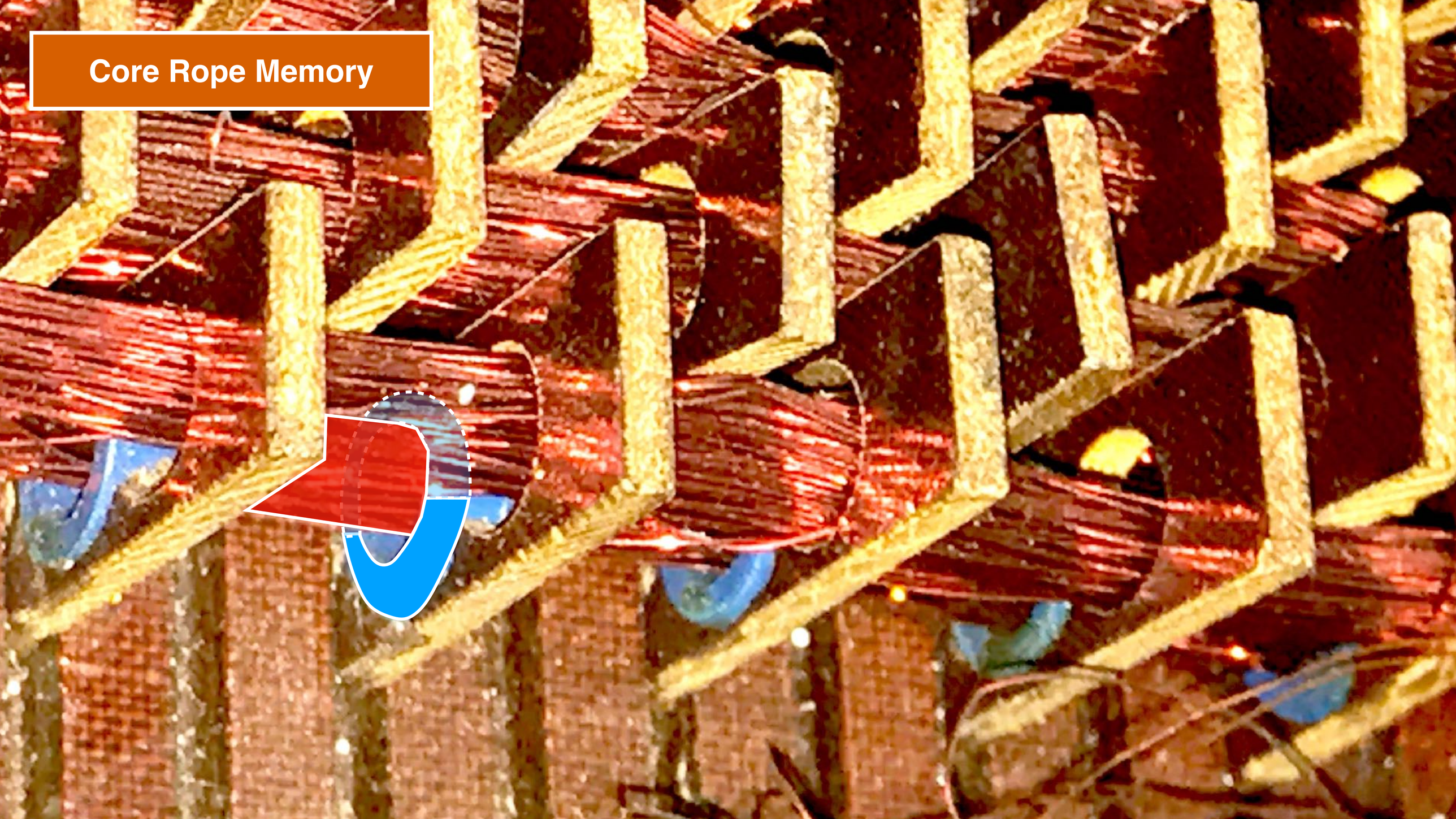
Core Rope Memory



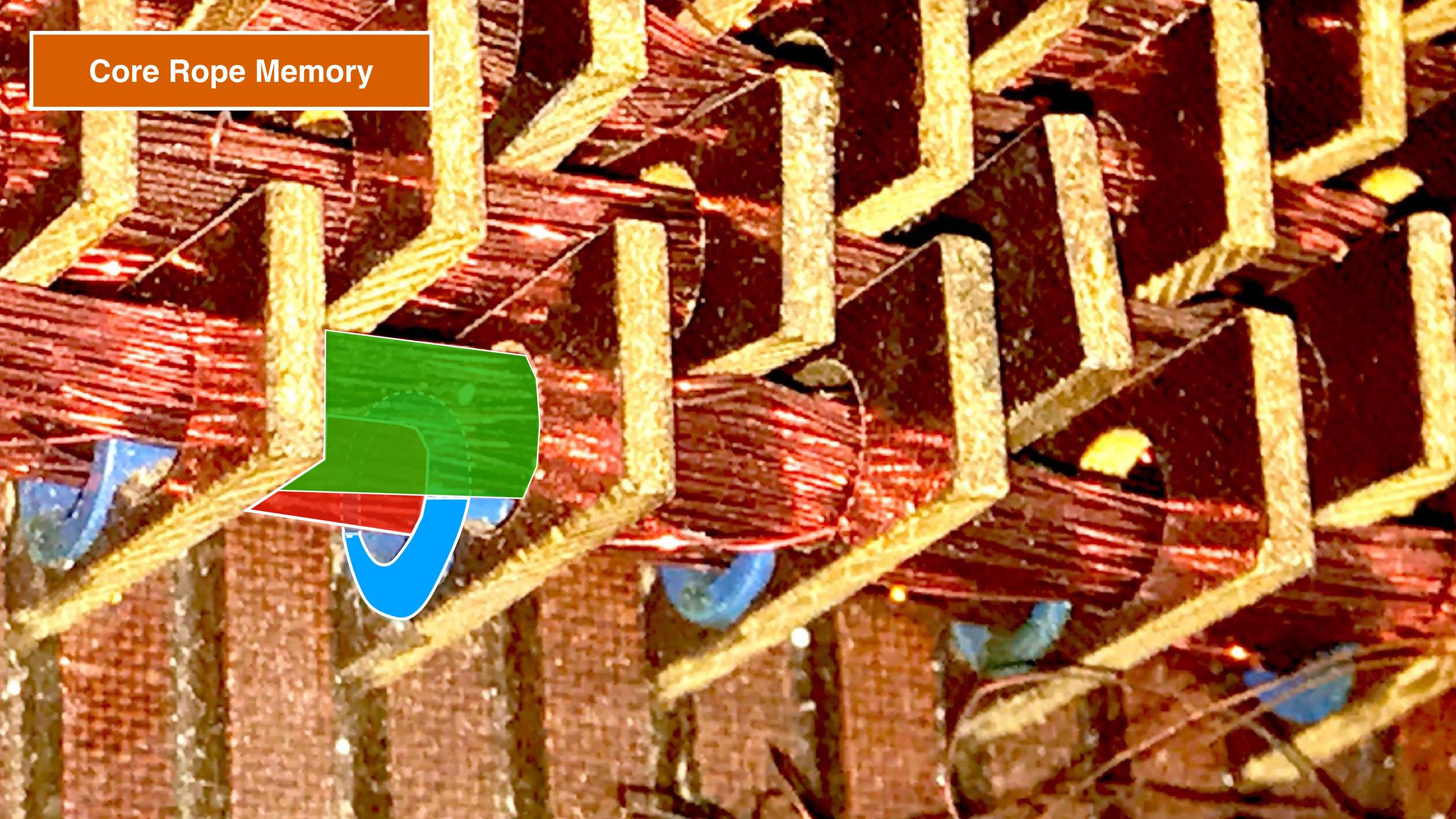
Core Rope Memory



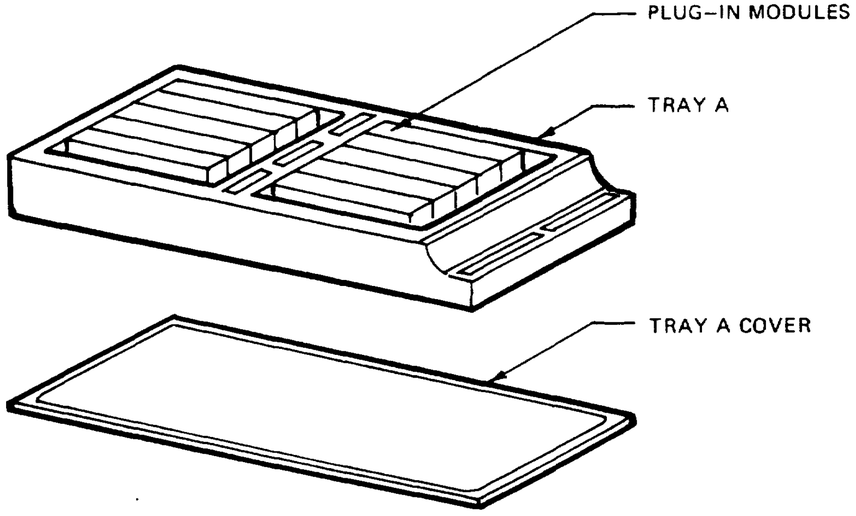
Core Rope Memory

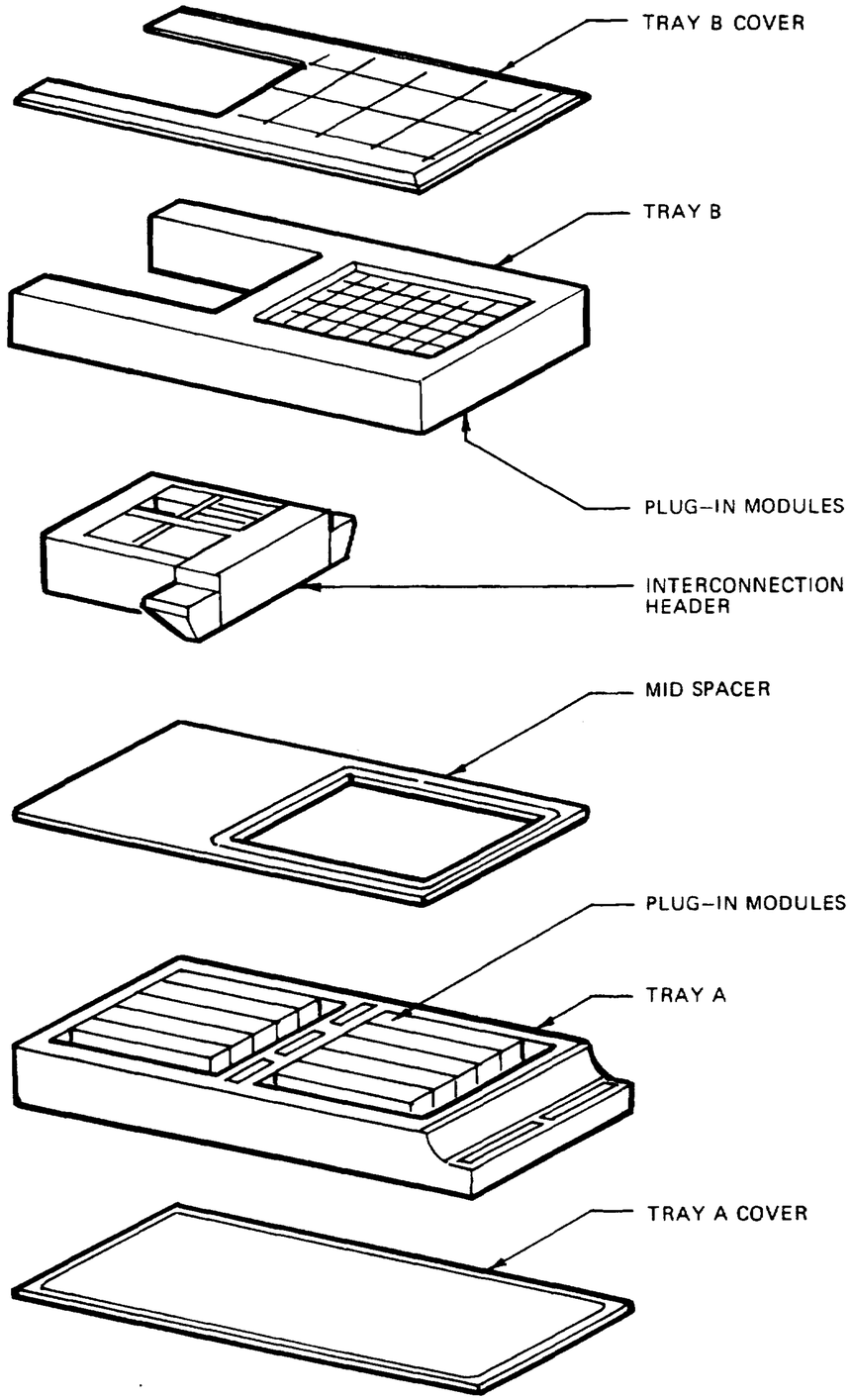


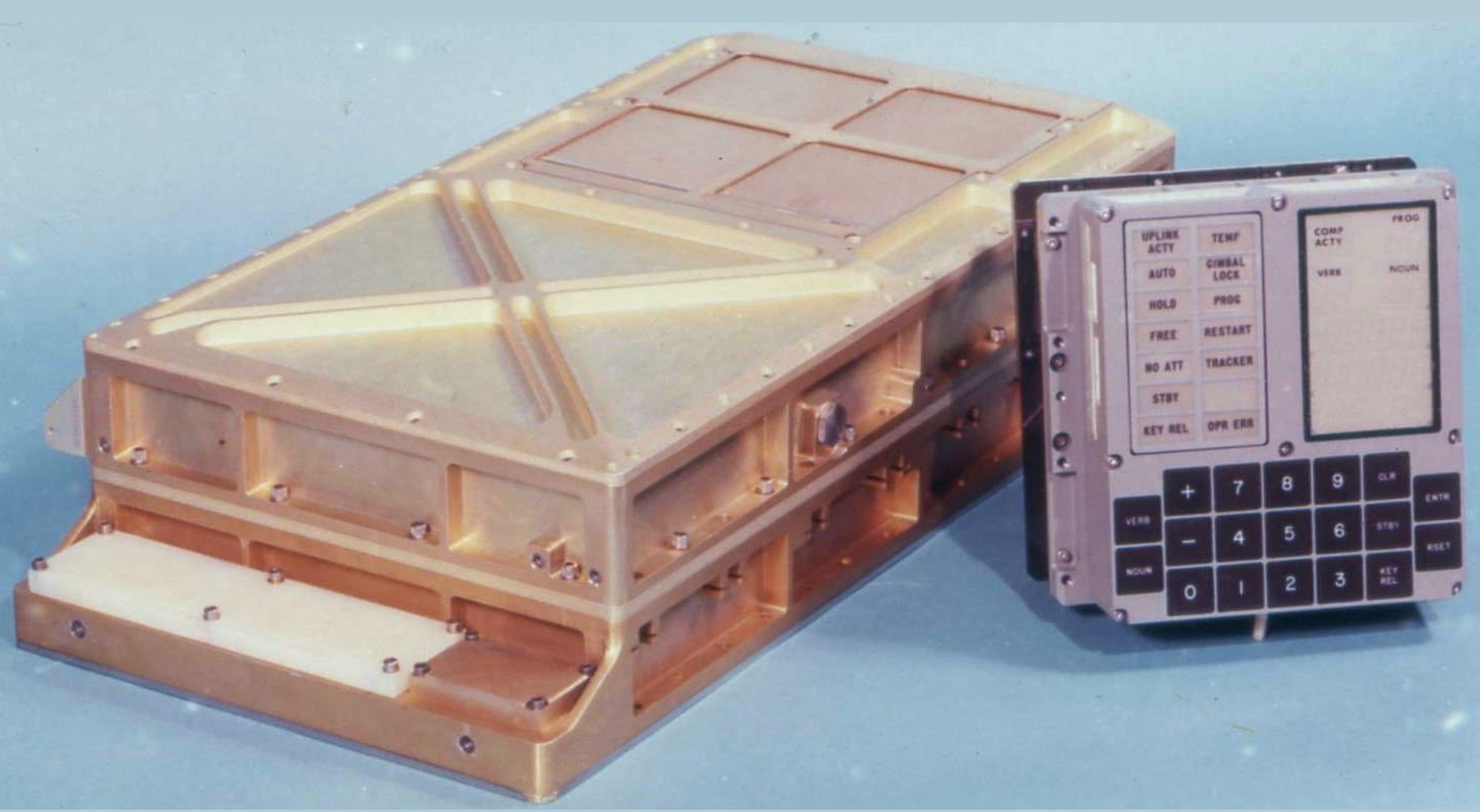
Core Rope Memory

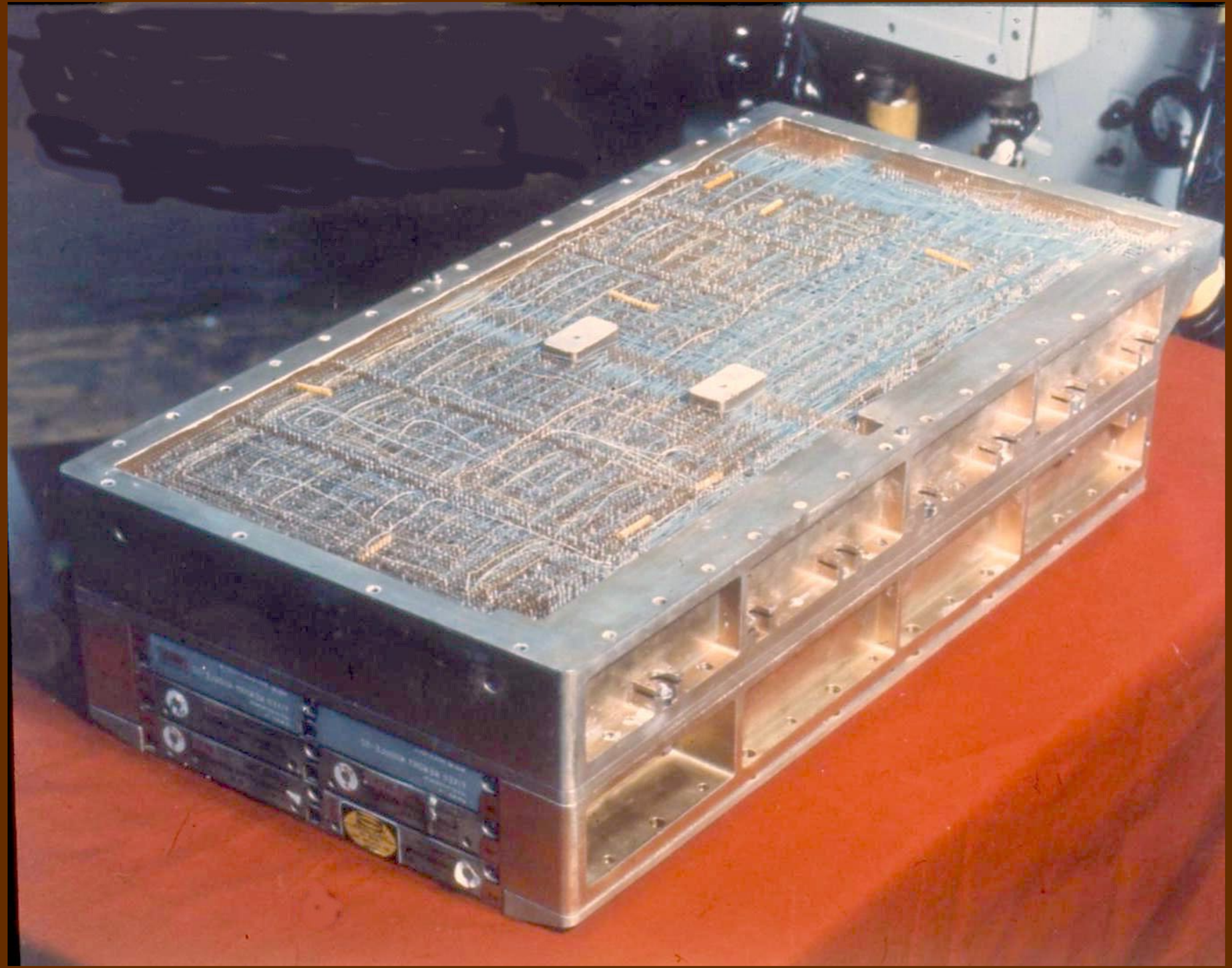


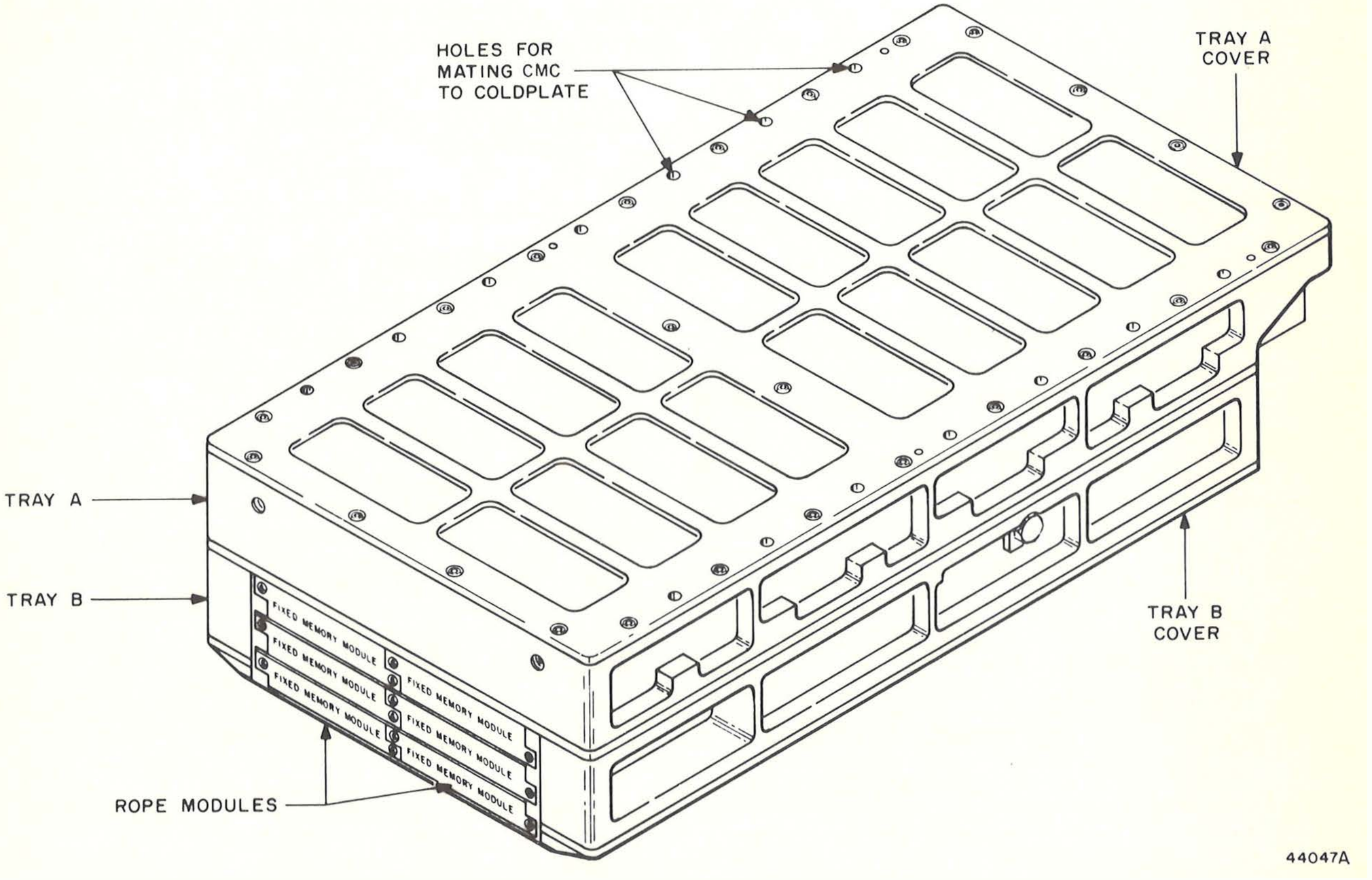
Core Rope Memory

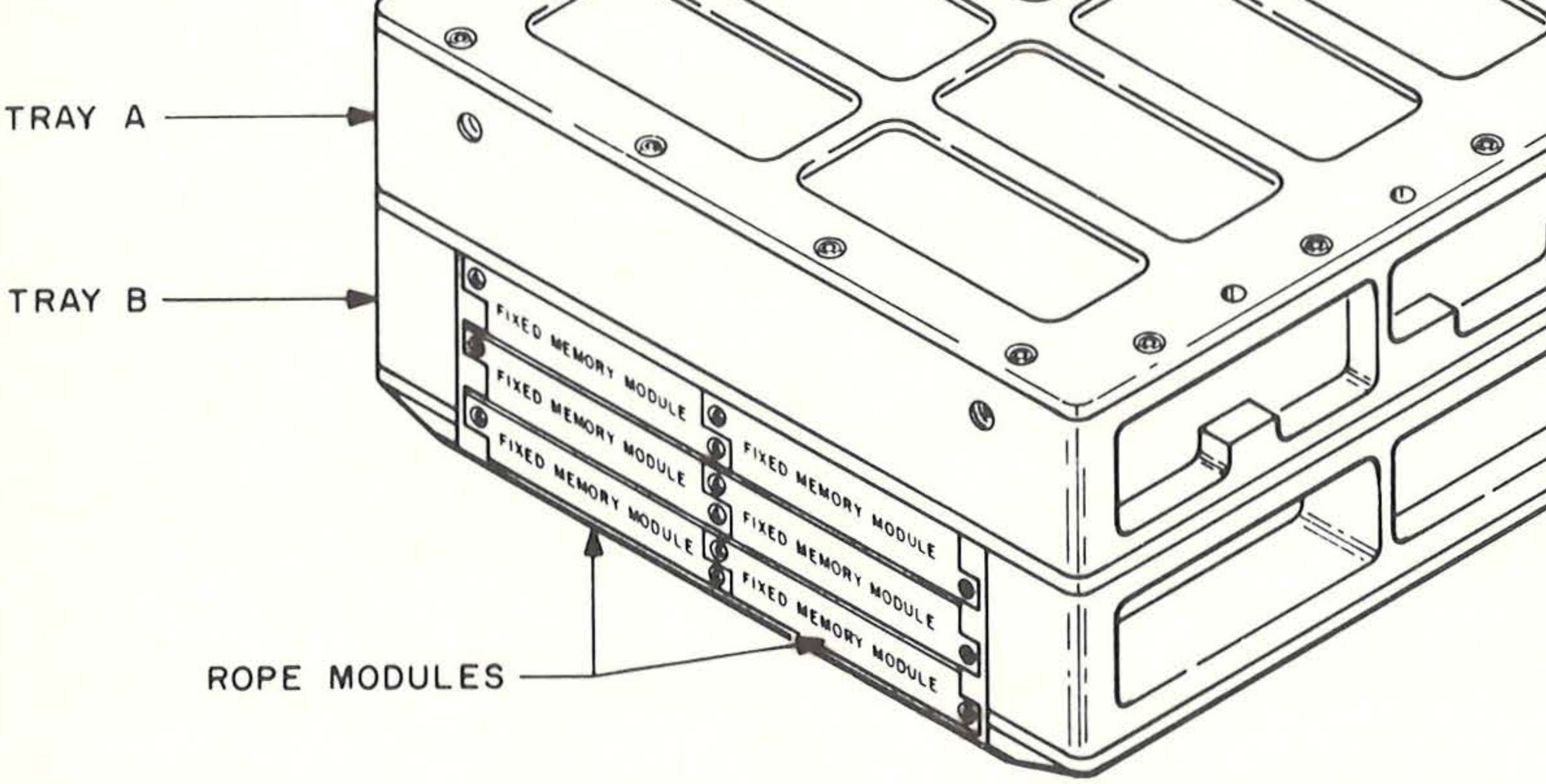

PLUG–IN MODULES
TRAY A
TRAY A COVER

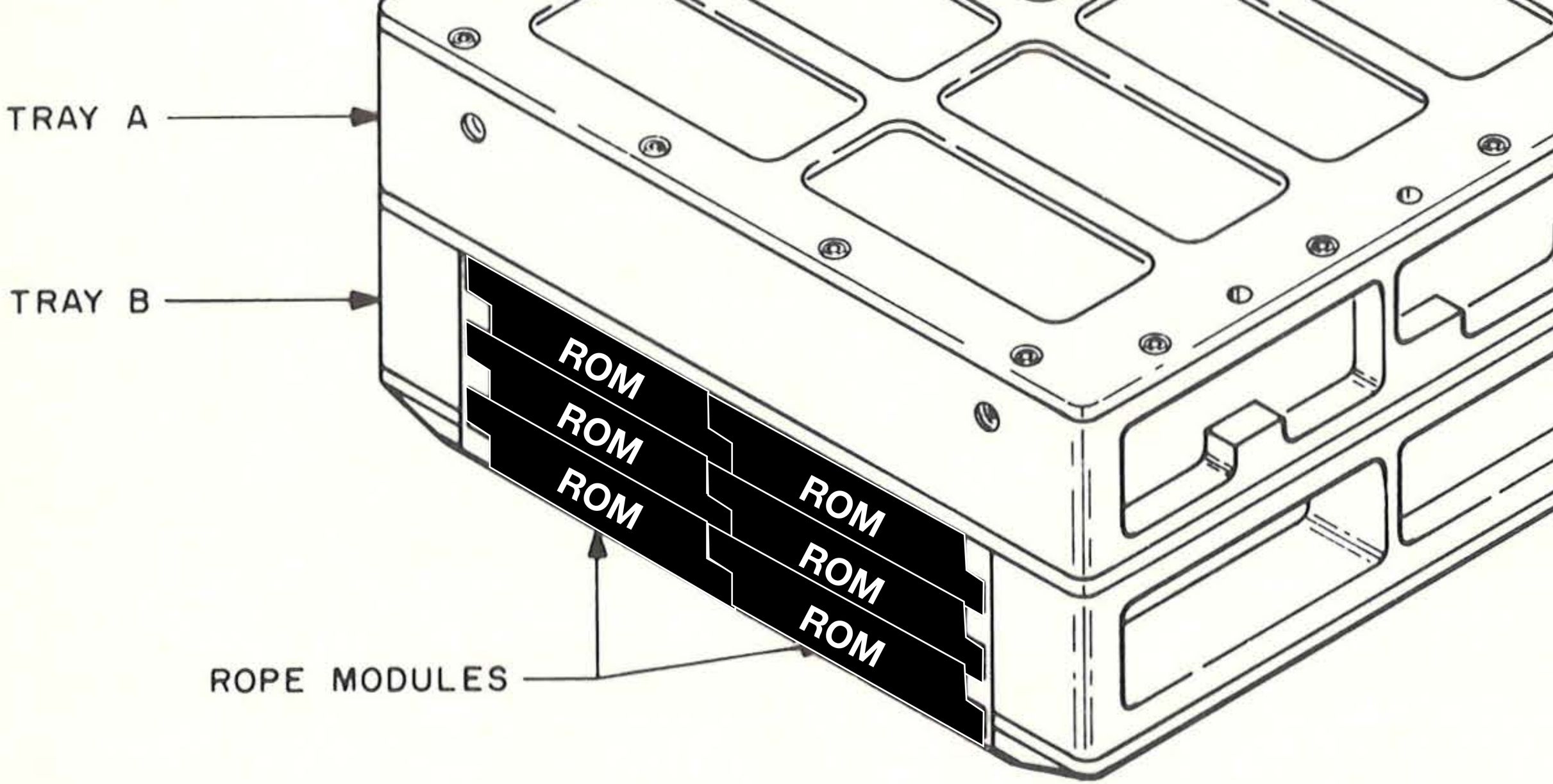

TRAY B COVER
TRAY B
PLUG–IN MODULES
INTERCONNECTION HEADER
MID SPACER
PLUG–IN MODULES
TRAY A
TRAY A COVER


UPLINK ACTY
TEMP
AUTO
GIMBAL LOCK
HOLD
PROG
FREE
RESTART
NO ATT
TRACKER
STBY
KEY REL
OPR ERR
COMP ACTY
PROG
VERB
NOUN
VERB
NOUN
+
7
8
9
CLR
ENTR
−
4
5
6
STBY
RSET
0
1
2
3
KEY REL




HOLES FOR
MATING CMC
TO COLDPLATE
TRAY A
COVER
TRAY A
TRAY B
TRAY B
COVER
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
ROPE MODULES
44047A


TRAY A
TRAY B
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
FIXED MEMORY MODULE
ROPE MODULES


TRAY A
TRAY B
ROM
ROM
ROM
ROM
ROM
ROM
ROPE MODULES

Hardware

- Timing
- Schematics
- Sequencer
- ICs
- Microcode
- Modules
- Unprogrammed Sequences
- Trays

Hardware

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

Peripherals

Peripherals

Peripherals

- Gyroscope
- Accelerometer
- Optics

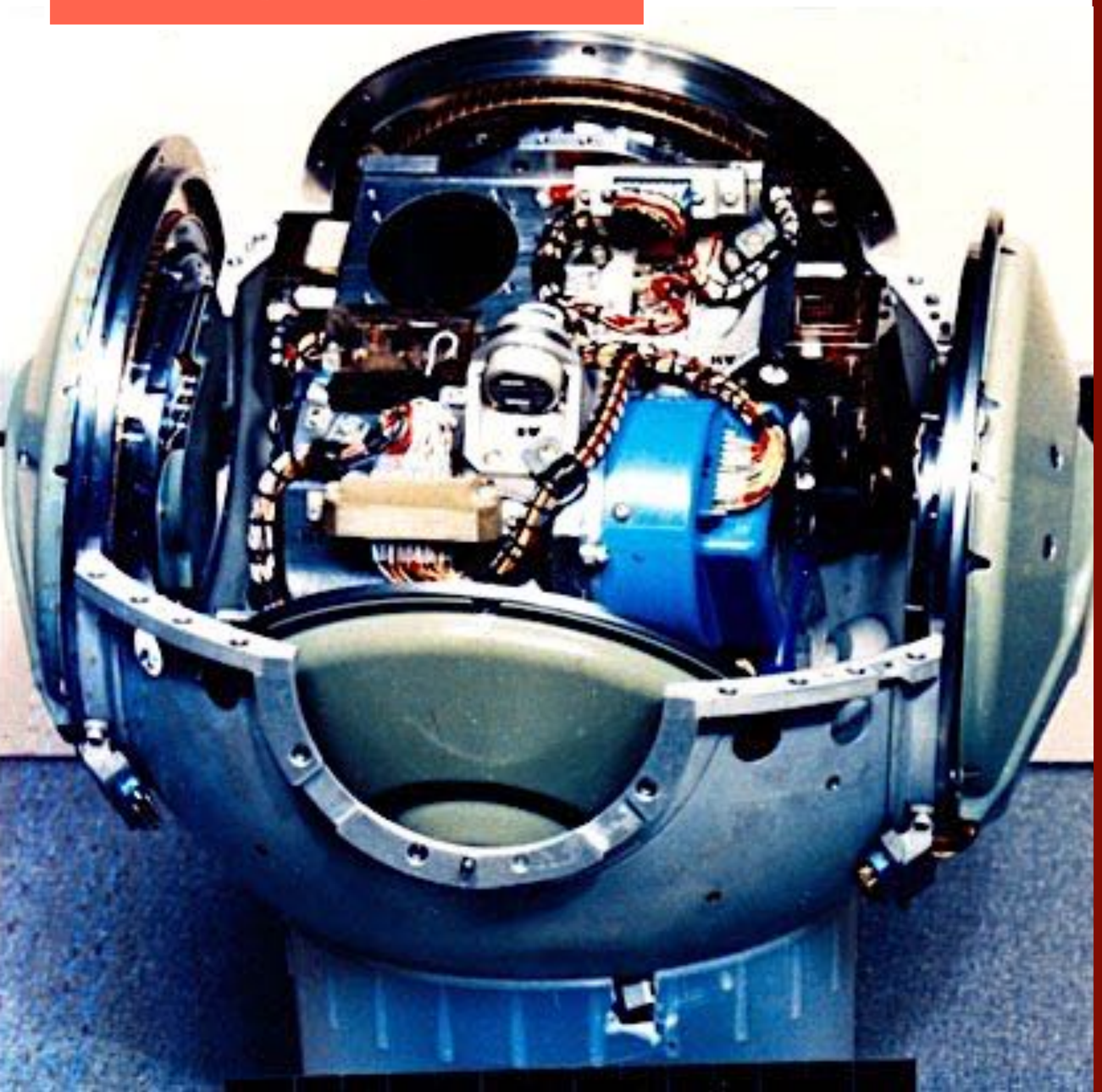
Peripherals

- Gyroscope
- Accelerometer
- Optics
- Landing Radar
- Rendezvous Radar
- RCS Jets

Peripherals

- Gyroscope
- Accelerometer
- Optics
- Landing Radar
- Rendezvous Radar
- RCS Jets
- DSKY
- Uplink/Downlink

Gyroscope



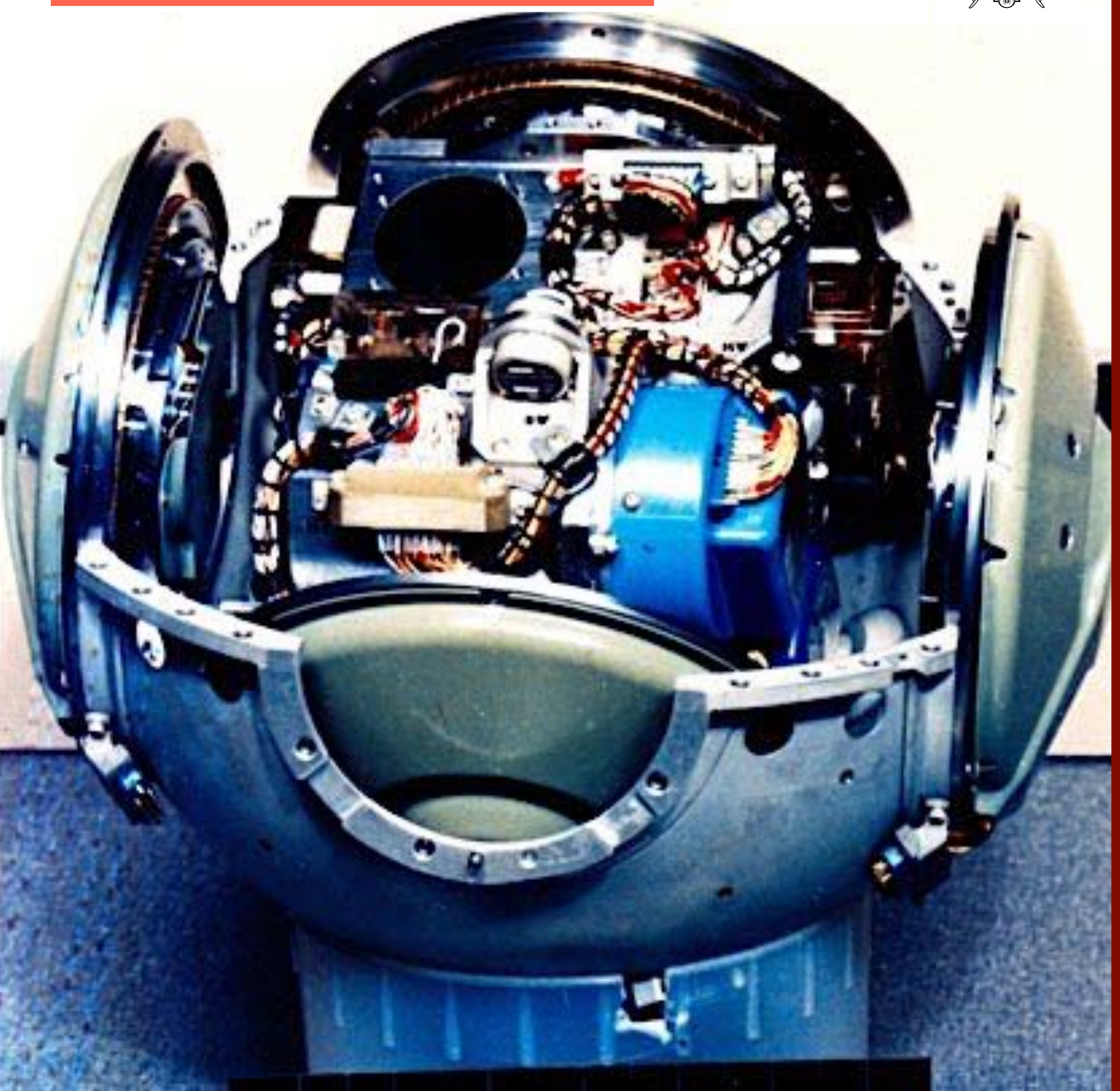
Counters

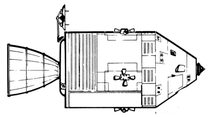
CDUX	01A	X gimbal
CDUY	01B	Y gimbal
CDUZ	01C	Z gimbal
CDUXCMD	028	X gimbal out
CDUYCMD	029	Y gimbal out
CDUZCMD	02A	Z gimbal out

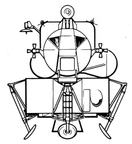
I/O

CHAN12	00A	3 coarse align enable 4 zero values 5 enable error ctr
CHAN14	00C	12-14 drive Z, Y, X
CHAN30	018	8 operate 10 cage 11 CDU fail 12 fail

Gyroscope







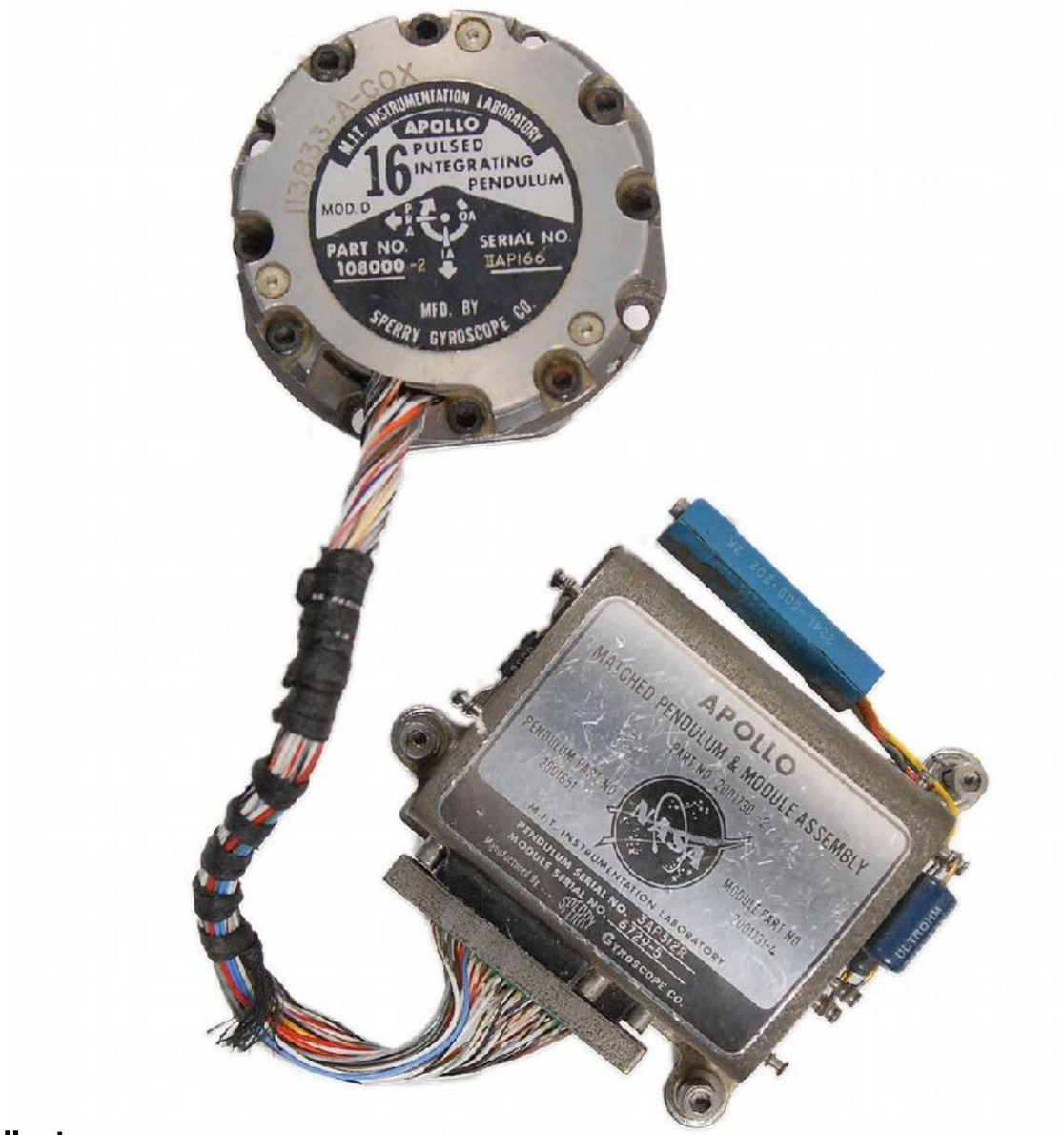
Counters

CDUX	01A	X gimbal
CDUY	01B	Y gimbal
CDUZ	01C	Z gimbal
CDUXCMD	028	X gimbal out
CDUYCMD	029	Y gimbal out
CDUZCMD	02A	Z gimbal out

I/O

CHAN12	00A	3 coarse align enable 4 zero values 5 enable error ctr
CHAN14	00C	12-14 drive Z, Y, X
CHAN30	018	8 operate 10 cage 11 CDU fail 12 fail

Accelerometer



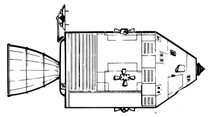
Counters

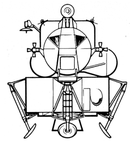
PIPAX	01D	value X
PIPAY	01E	value Y
PIPAZ	026	value Z

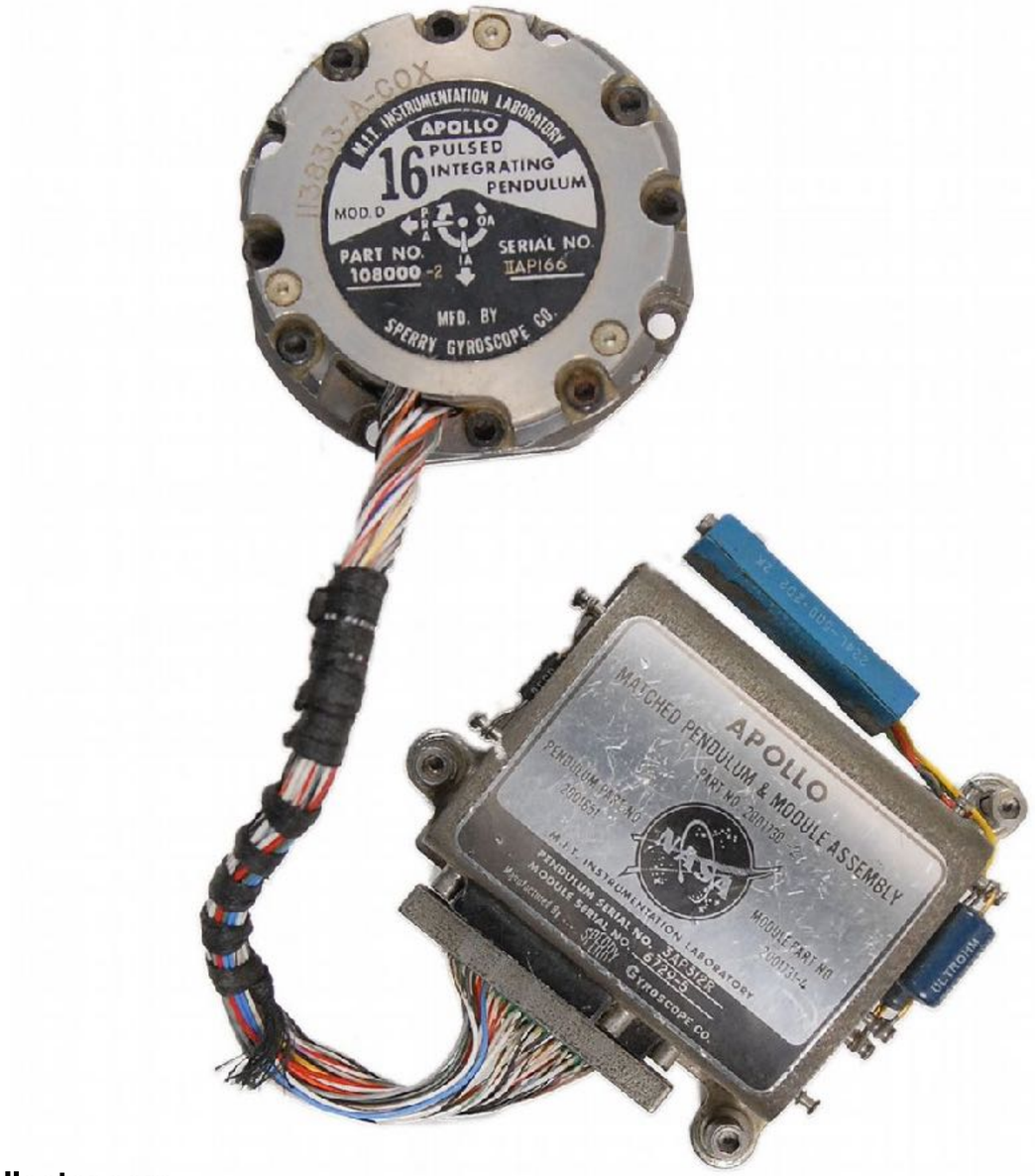
I/O

CHAN33	01B	12 fail

Accelerometer







Counters

PIPAX 01D value X

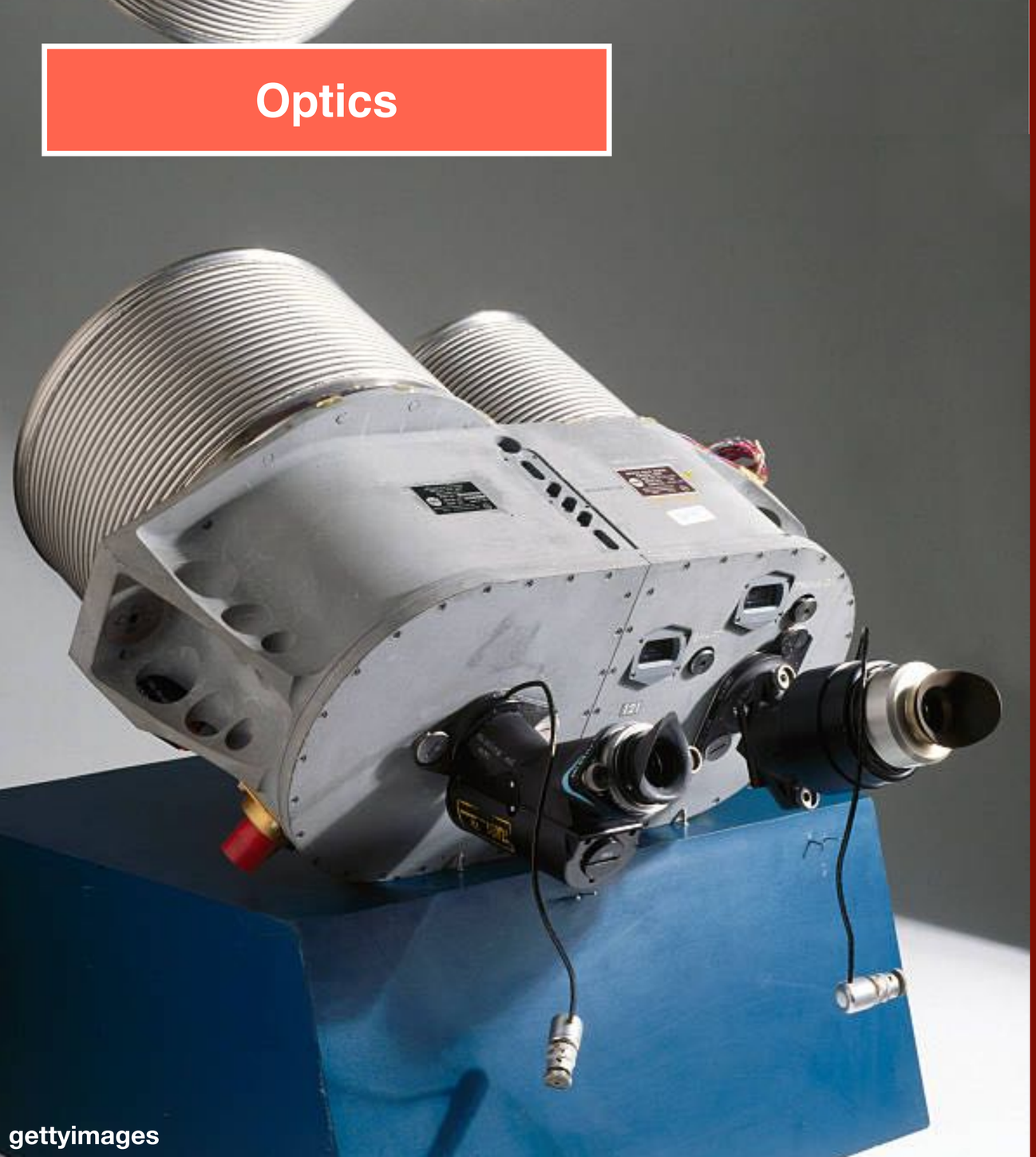
PIPAY 01E value Y

PIPAZ 026 value Z

I/O

CHAN33 01B 12 fail

Optics



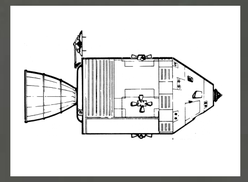
Counters

OPTY	01D	y axis in
OPTX	01E	x axis in
OPTYCMD	02B	y axis out
OPTXCMD	02C	x axis out

I/O

CHAN12	00A	0 zero CDU 1 enable error ctr 9 zero optics 10 disengage DAC
CHAN30	018	6 CDU fail
CHAN33	01B	3 zero optics

Optics



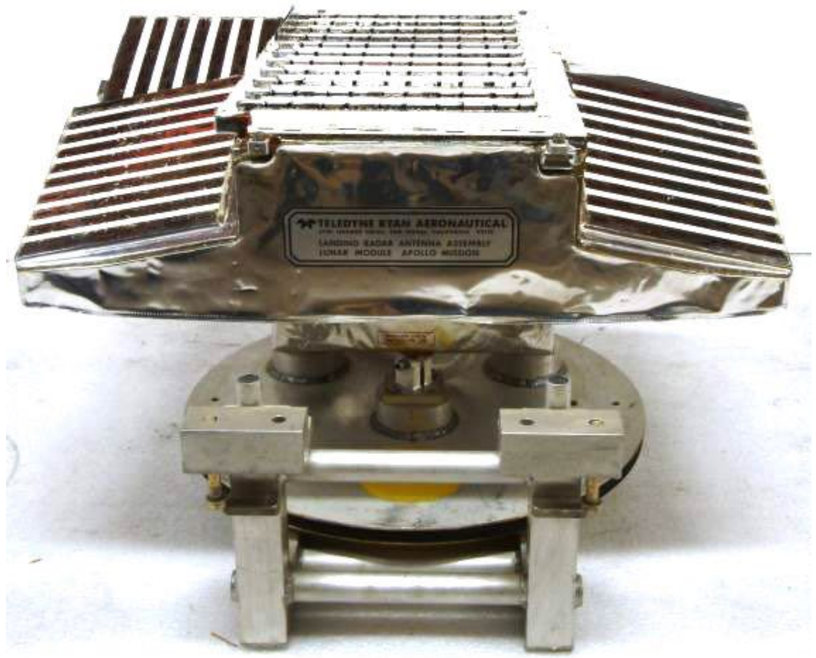
Counters

OPTY	01D	y axis in
OPTX	01E	x axis in
OPTYCMD	02B	y axis out
OPTXCMD	02C	x axis out

I/O

CHAN12	00A	0 zero CDU 1 enable error ctr 9 zero optics 10 disengage DAC
CHAN30	018	6 CDU fail
CHAN33	01B	3 zero optics

Landing Radar



Smithsonian National Air and Space Museum

Interrupts

RADARUPT ▶ 428 — data ready

Counters

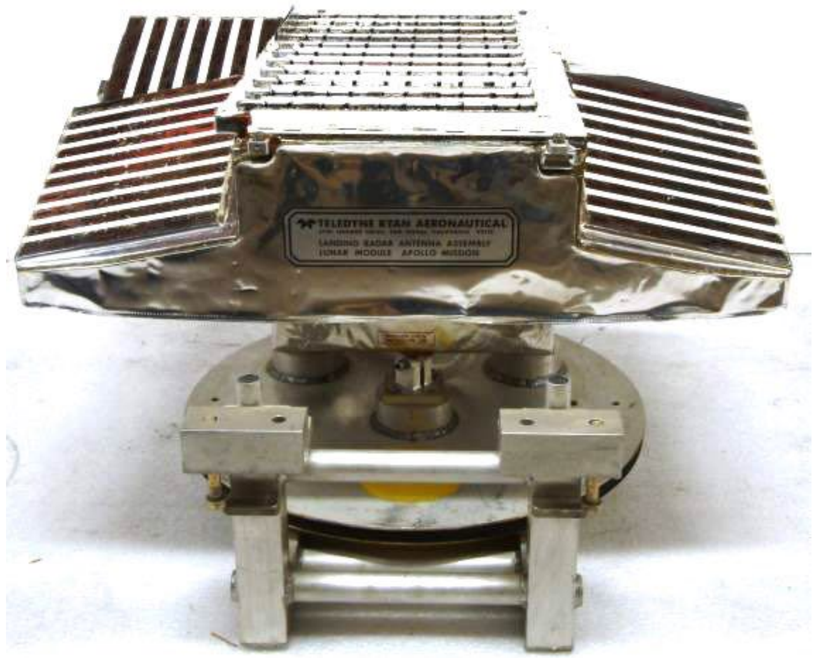
RNRAD 026 — data

I/O

CHAN13 00B
0-2 radar select
3 activity

CHAN33 01B
1 auto-power on
2 range low scale
3 data good

Landing Radar



Smithsonian National Air and Space Museum

Interrupts

RADARUPT ▶ 428 — data ready

Counters

RNRAD 026 — data

I/O

CHAN13 00B
- 0-2 radar select
- 3 activity
- 1 auto-power on

CHAN33 01B
- 2 range low scale
- 3 data good

Rendezvous Radar

Interrupts

RADARUPT ▶ 428 — data ready

Counters

CDUT	01D	trunnion angle
CDUS	01E	shaft angle
RNRAD	026	data

I/O

CHAN12	00A	0 zero angles 1 enable error ctr 13 enbl auto track
CHAN13	00B	0-2 radar select 3 activity
CHAN30	018	6 CDU fail
CHAN33	01B	1 auto-power on 2 range low scale 3 data good

Rendezvous Radar

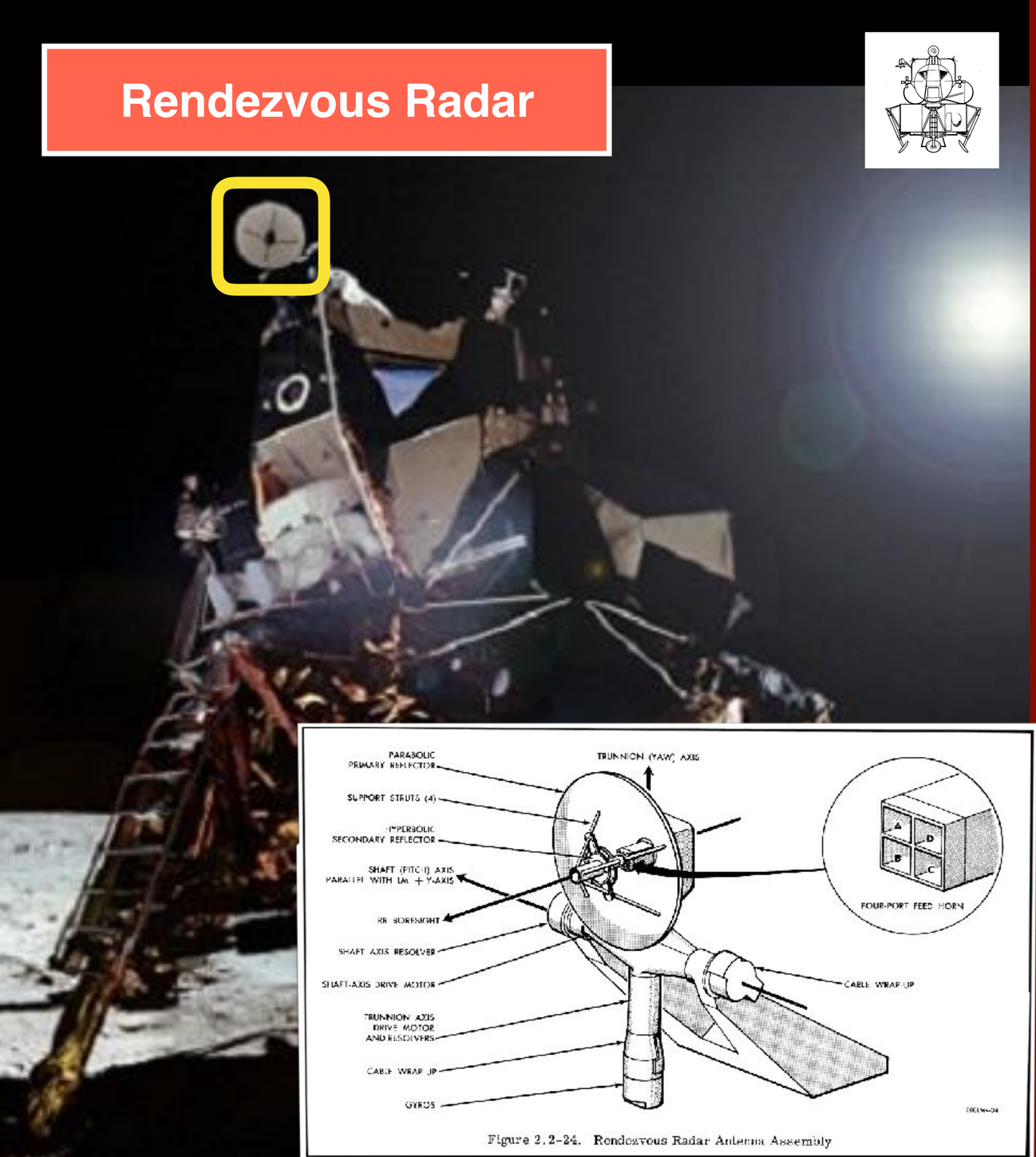


Figure 2.2-24. Rendezvous Radar Antenna Assembly

Interrupts

RADARUPT ▶ 428 — data ready

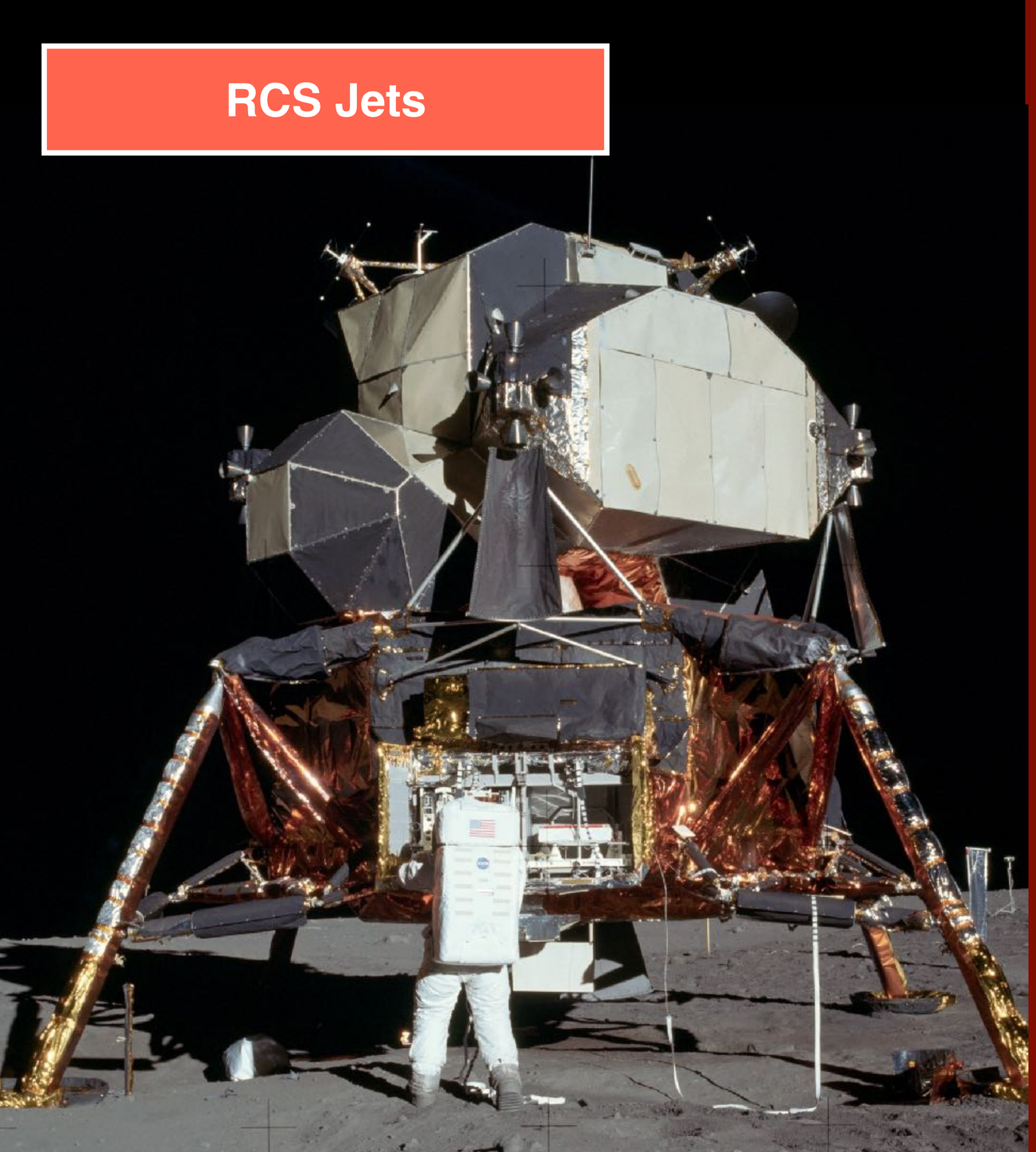
Counters

CDUT	01D	trunnion angle
CDUS	01E	shaft angle
RNRAD	026	data

I/O

CHAN12	00A	0 zero angles 1 enable error ctr 13 enbl auto track
CHAN13	00B	0-2 radar select 3 activity
CHAN30	018	6 CDU fail
CHAN33	01B	1 auto-power on 2 range low scale 3 data good

RCS Jets



Interrupts

T6RUPT ▶ 404 — TIME6 reached 0

Counters

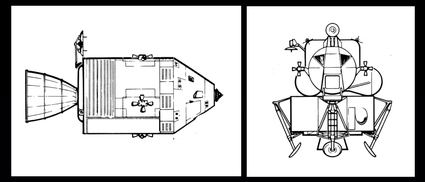
TIME6 019 — timer

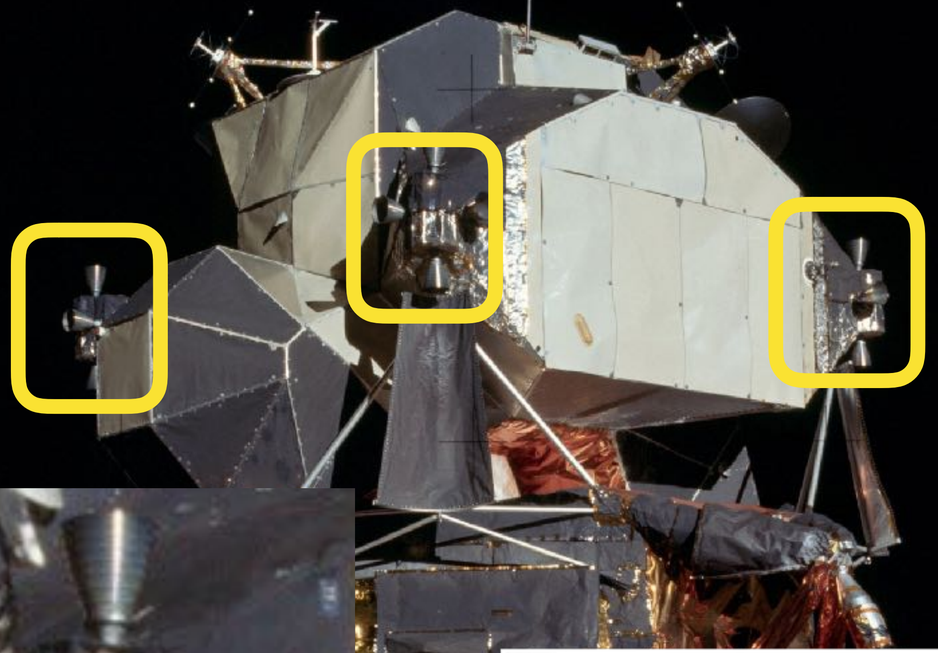
I/O

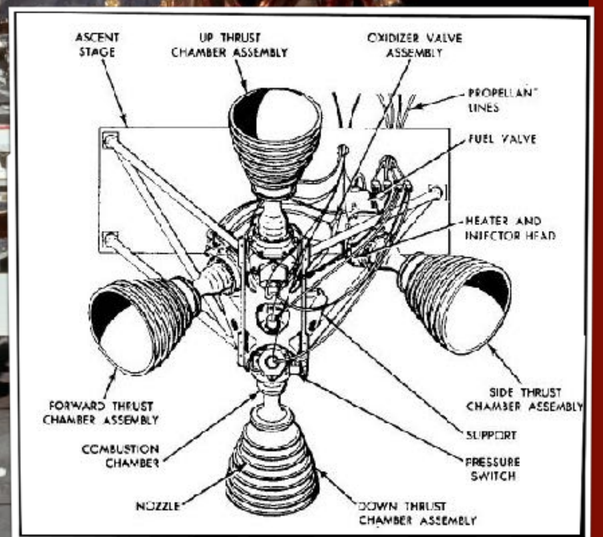
CHAN5 005 — 0-7 one bit per jet

CHAN6 006 — 0-7 one bit per jet

RCS Jets







Interrupts

T6RUPT ▶ 404 — TIME6 reached 0

Counters

TIME6 019 — timer

I/O

CHAN5 005 — 0-7 one bit per jet

CHAN6 006 — 0-7 one bit per jet

DSKY

Interrupts

KEYRUPT ▶ 414 key pressed

I/O

OUTO 008 numeric display
0-10 bits in this row
11-14 row

DSALMOUT 009 1-6 lamps

MNKEYIN 00D 0-4 key code

The Command Module contains two DSKYs,
so it has a second set of registers.

DSKY

Interrupts

KEYRUPT ▶ 414 — key pressed

I/O

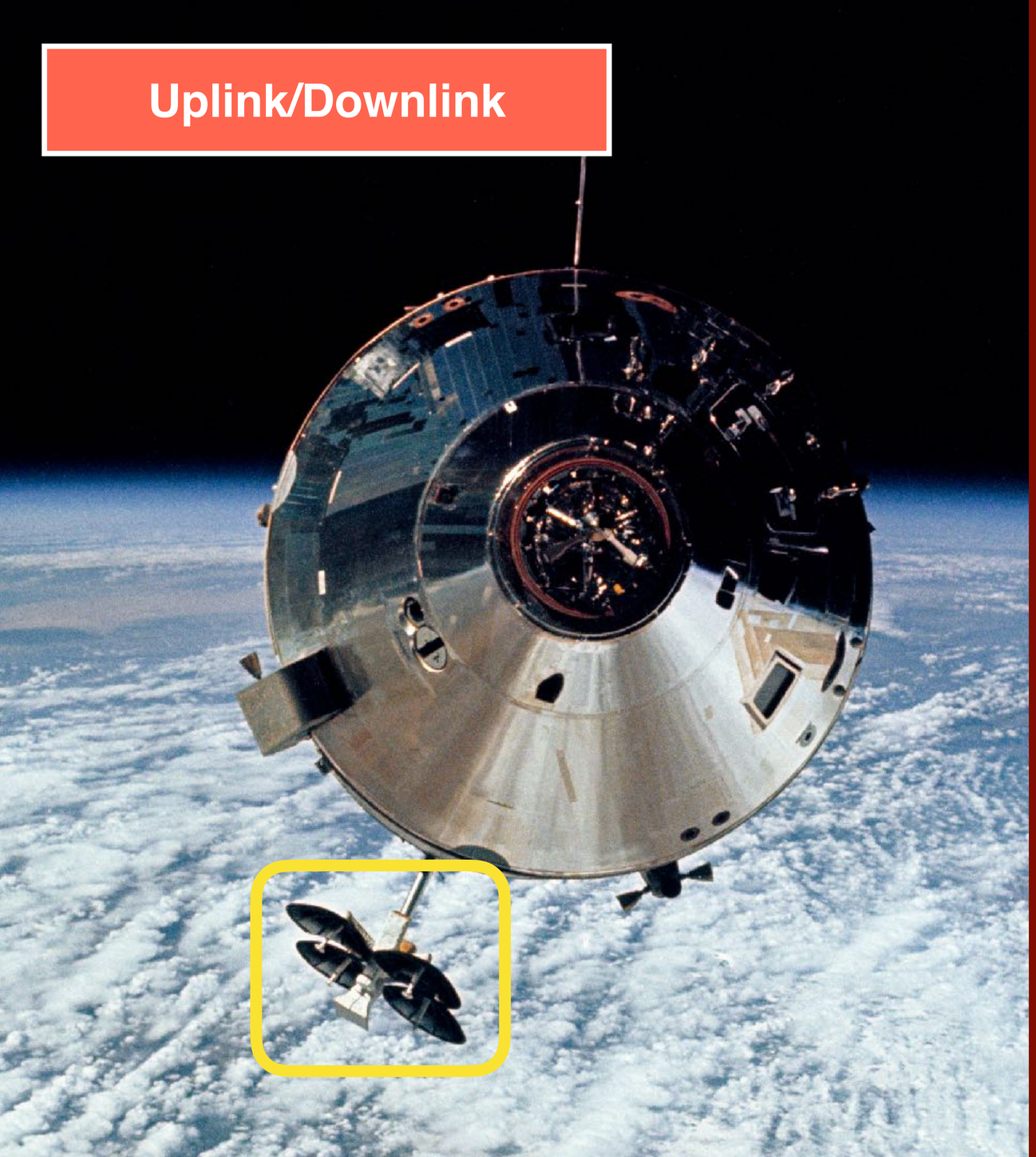
OUTO 008 — numeric display
0-10 bits in this row
11-14 row

DSALMOUT 009 — 1-6 lamps

MNKEYIN 00D — 0-4 key code

The Command Module contains two DSKYs, so it has a second set of registers.

Uplink/Downlink



Interrupts

UPRUPT	41C	data ready
DOWNRUPT	420	ready to send

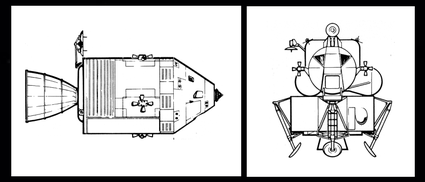
Counters

INLINK	025	uplink data

I/O

CHAN13	00B	6 dl word order
CHAN33	01B	9 block up input 10 up too fast 11 dl too fast
DNTM1	01B	downlink data

Uplink/Downlink



Interrupts

UPRUPT	41C	data ready
DOWNRUPT	420	ready to send

Counters

INLINK	025	uplink data

I/O

CHAN13	00B	6 dl word order
CHAN33	01B	9 block up input 10 up too fast 11 dl too fast
DNTM1	01B	downlink data

Peripherals

- Gyroscope
- Accelerometer
- Optics
- Landing Radar
- Rendezvous Radar
- RCS Jets
- DSKY
- Uplink/Downlink

Peripherals

Apollo Guidance Computer

Architecture

Mission Software

Hardware

Peripherals

System Software

System Software

System Software

System Software

- Priority-Based Cooperative
- Preemptive Real-Time
- Interactive
- Fault-Tolerant
- Virtual Machines

System Software

System Software

- Multitasking
- Interpreter
- Device Drivers
- Waitlist

System Software

- Multitasking
- Shell
- Interpreter
- Fault Recovery
- Device Drivers
- Waitlist

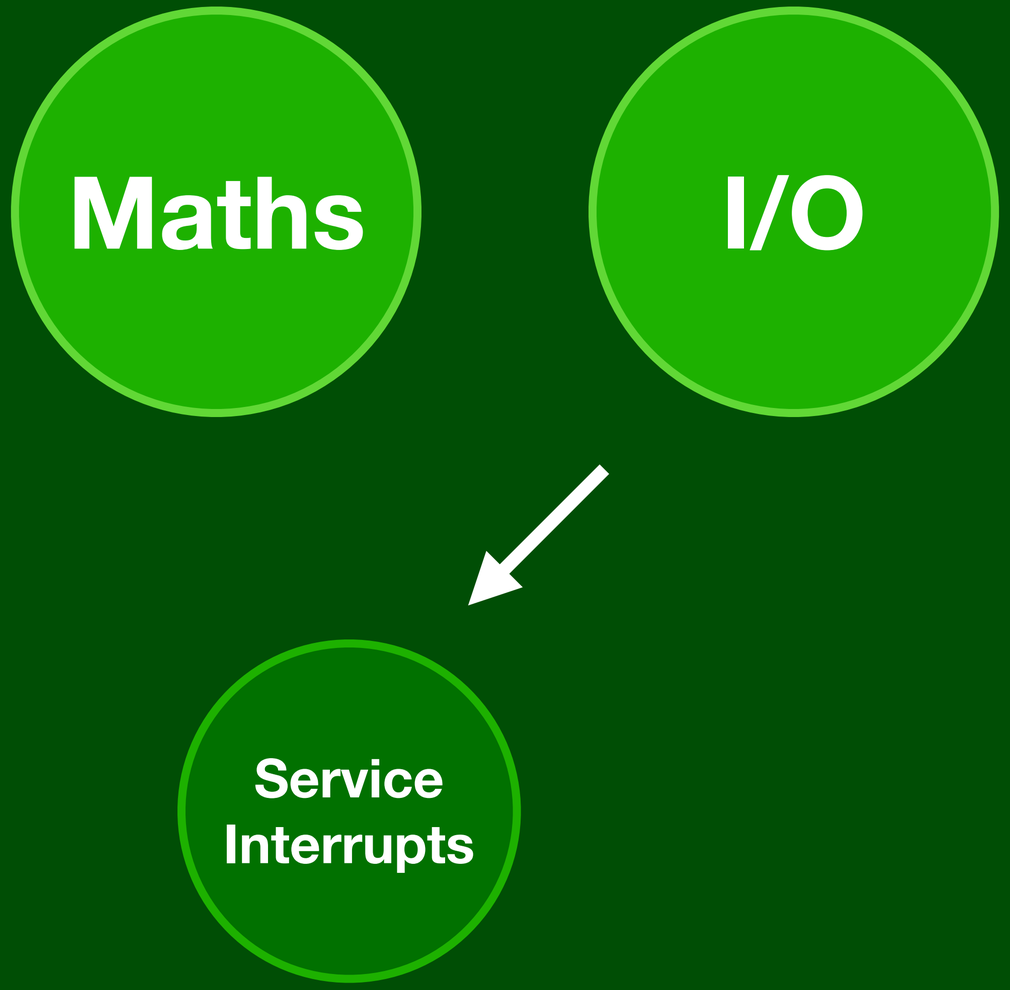
Multitasking

Multitasking

Maths

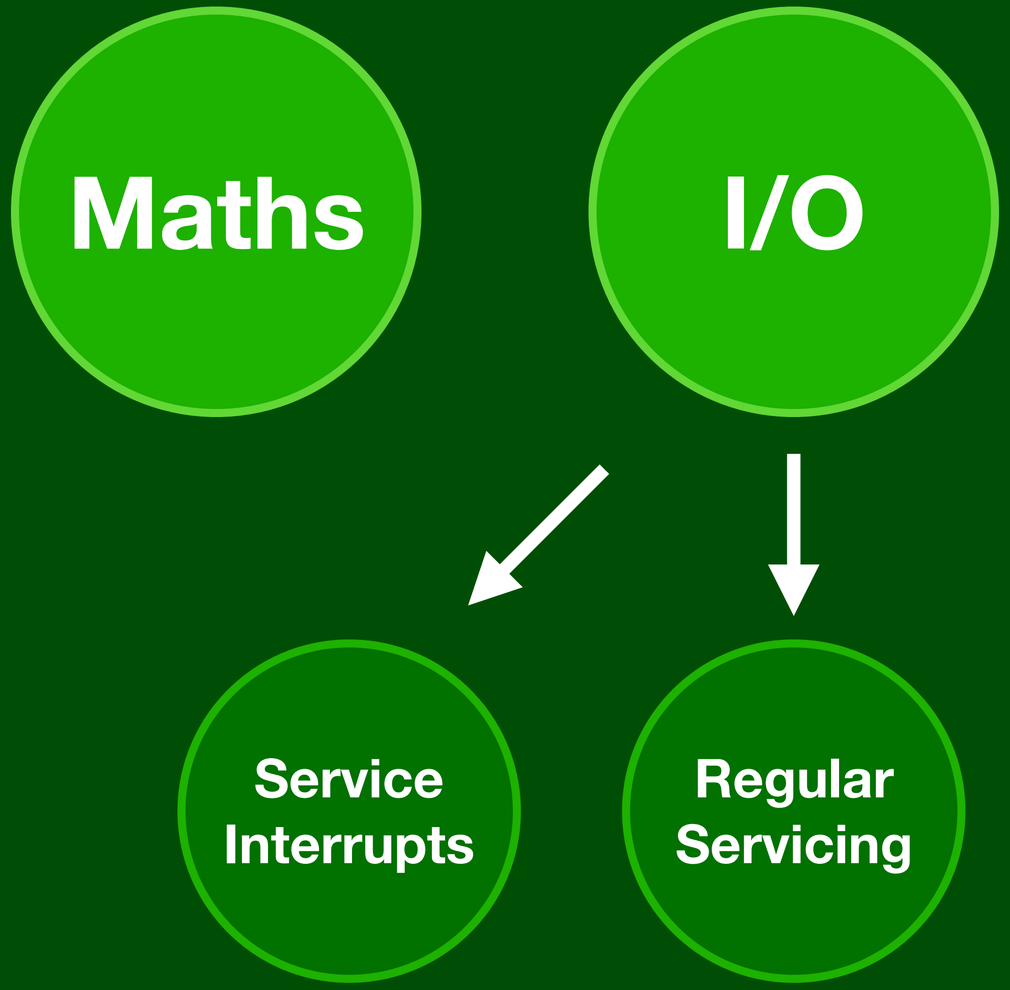
Multitasking

Multitasking



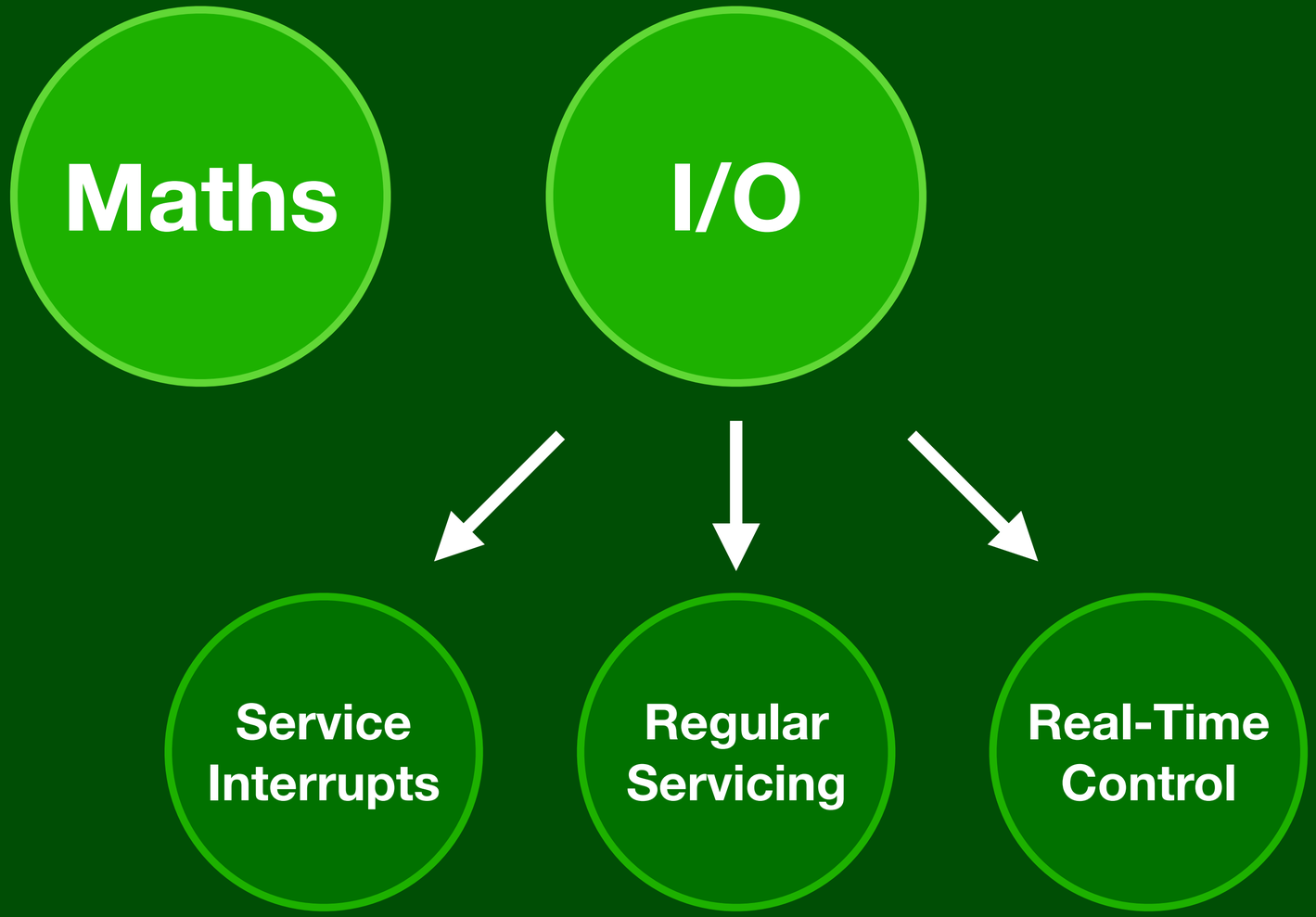
DSKY: Keypress

Multitasking



DSKY: Keypress —— Update Display

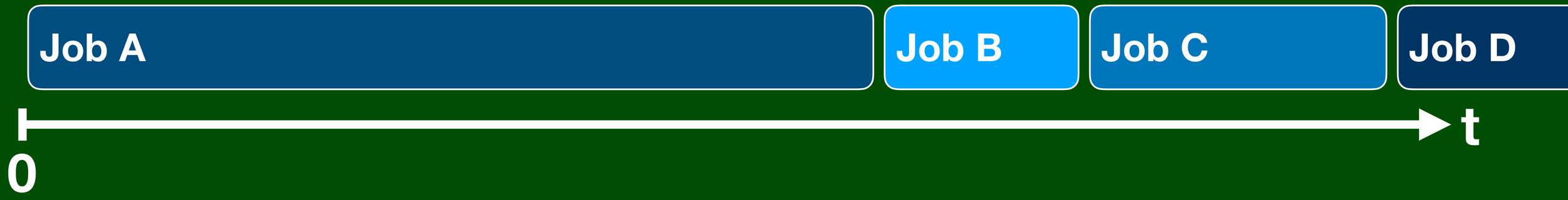
Multitasking



Scheduling

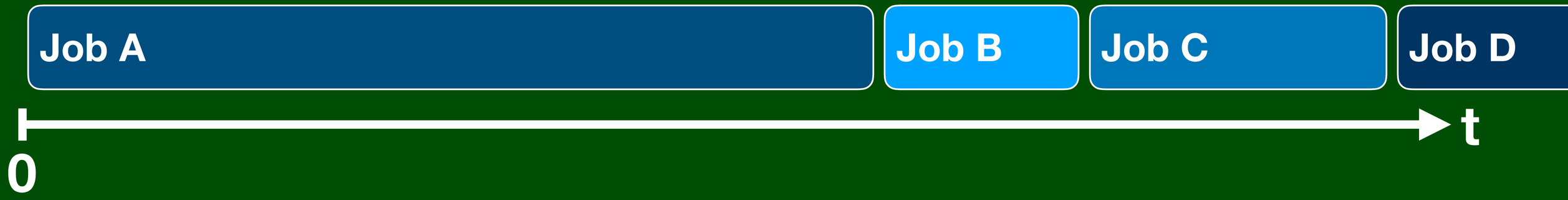
Scheduling

Scheduling



Scheduling

4 Highest
3 High
2 Medium
1 Low



Priority Scheduling

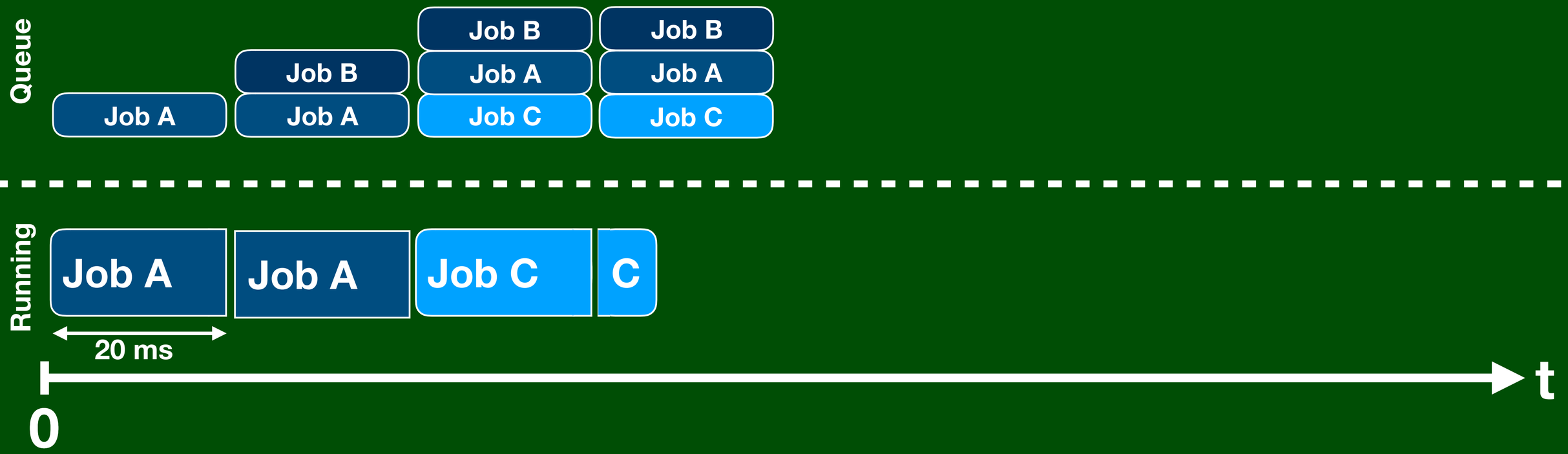
4 Highest
3 High
2 Medium
1 Low

Priority Scheduling

4 Highest
3 High
2 Medium
1 Low

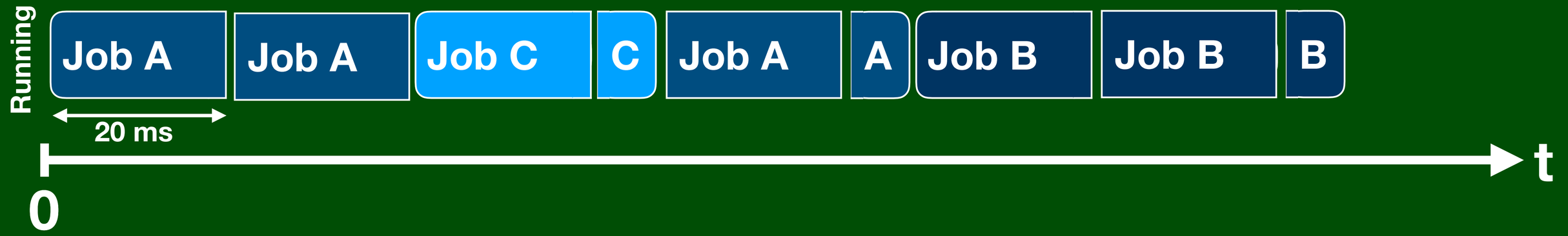
Priority Scheduling

4 Highest
3 High
2 Medium
1 Low



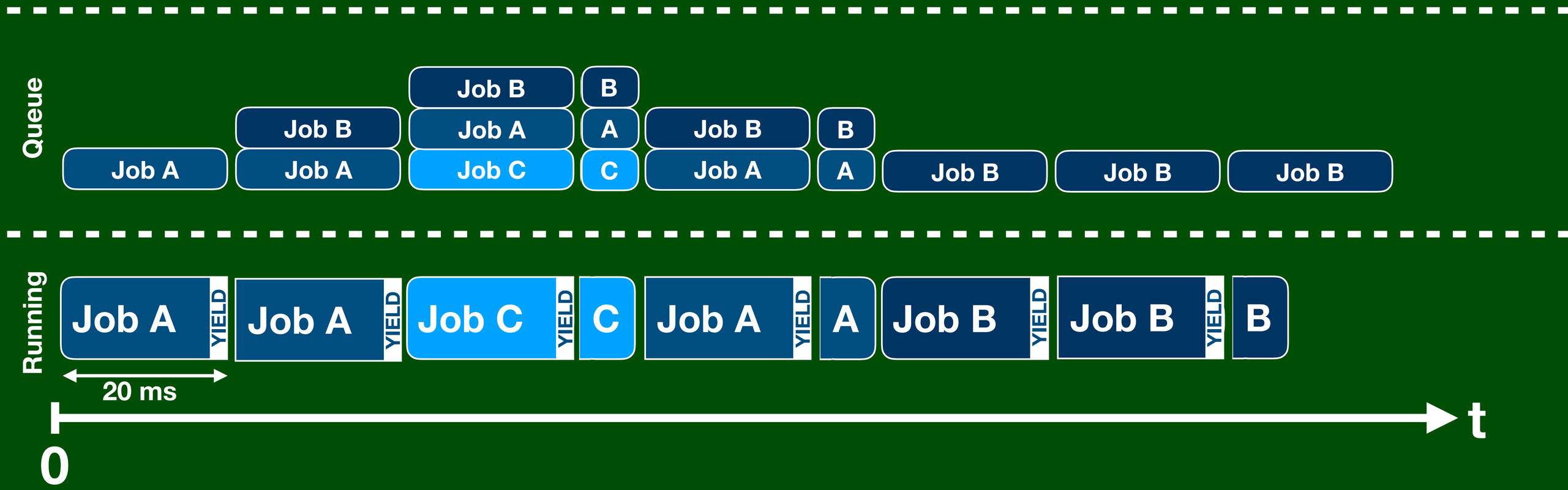
Priority Scheduling

4 Highest
3 High
2 Medium
1 Low



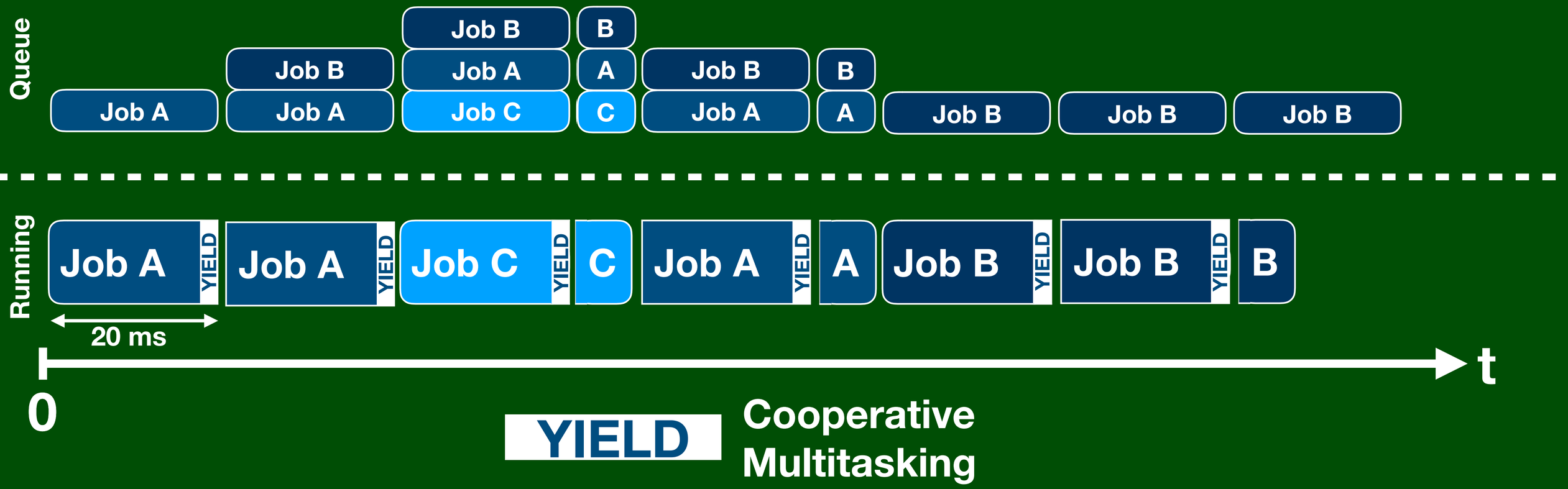
Priority Scheduling

4 Highest
3 High
2 Medium
1 Low



Priority Scheduling

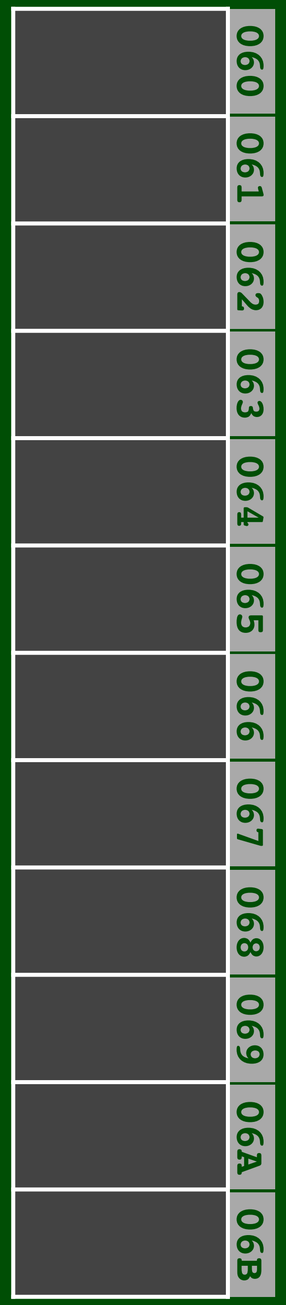
4 Highest
3 High
2 Medium
1 Low



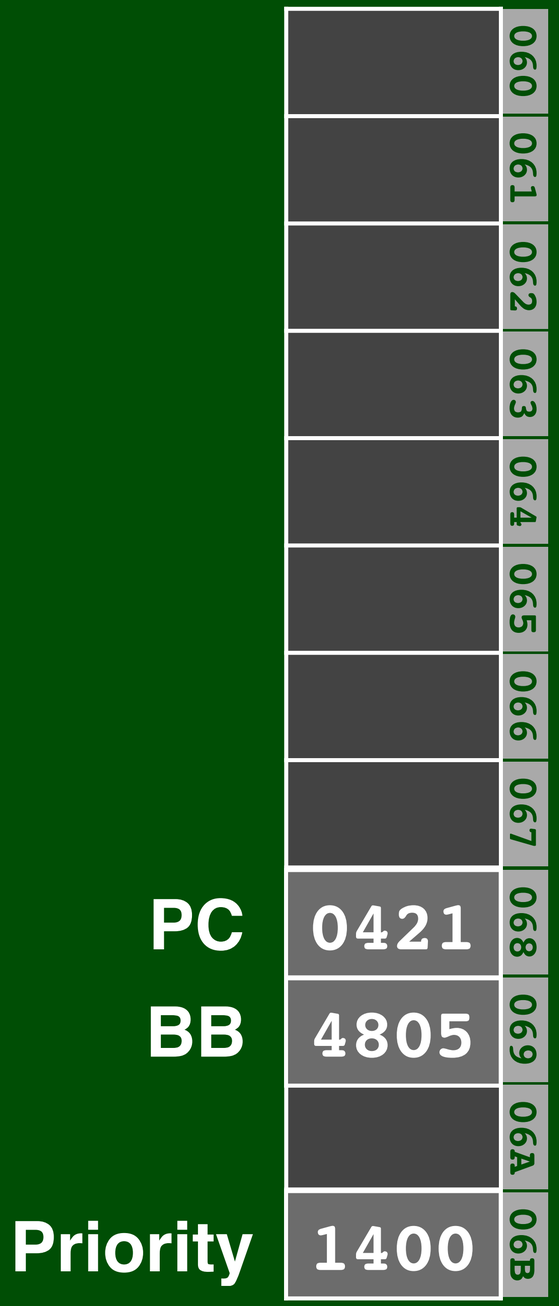
YIELD

Cooperative Multitasking

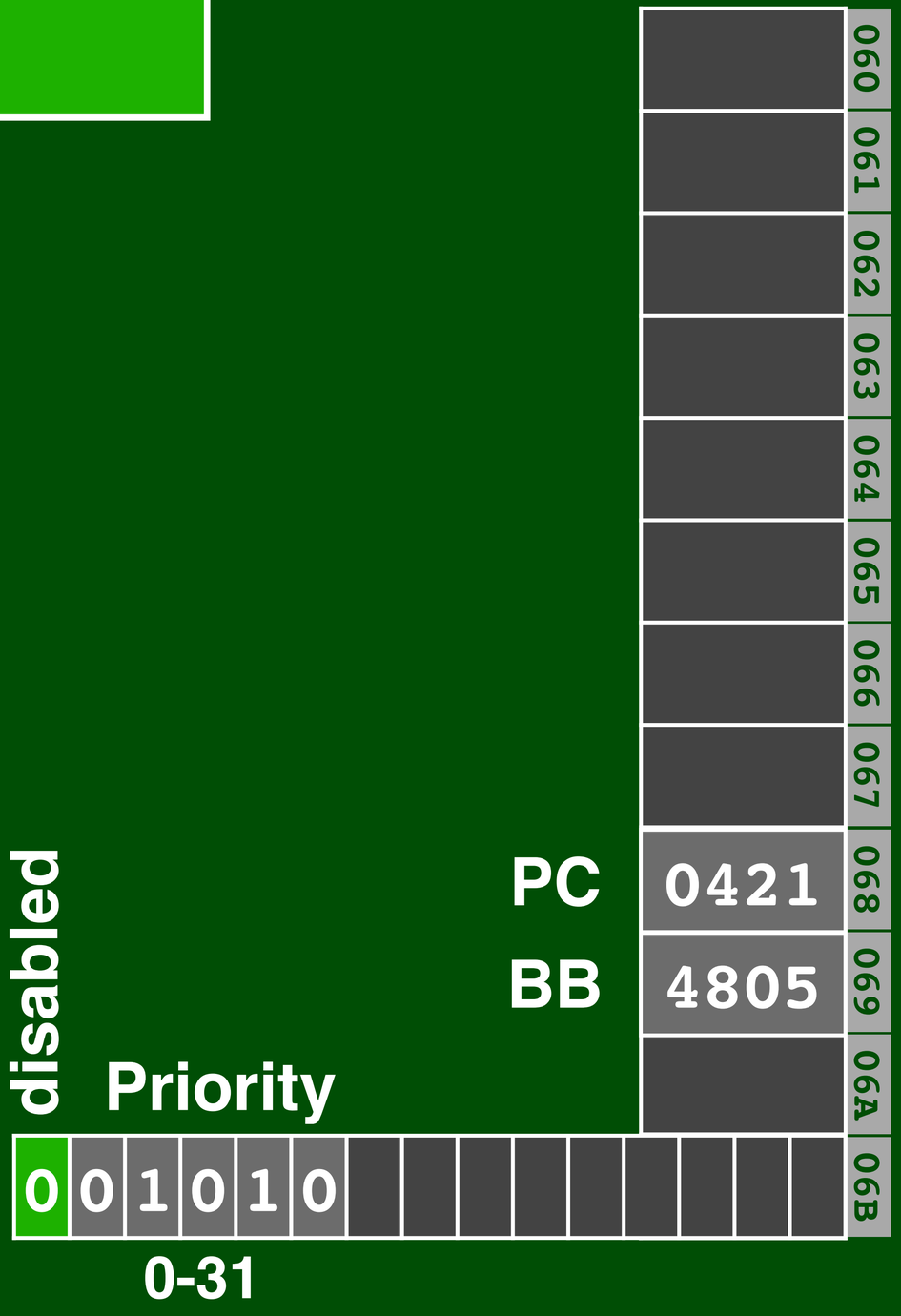
Job Entry



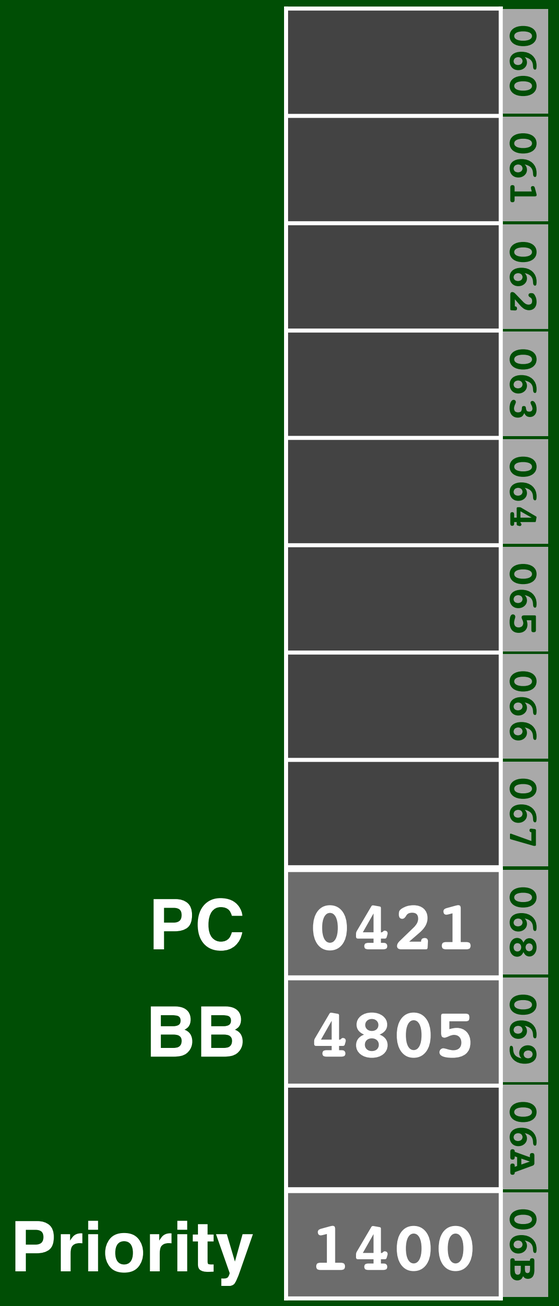
Job Entry


060
061
062
063
064
065
066
067
PC
0421
068
BB
4805
069
06A
Priority
1400
06B

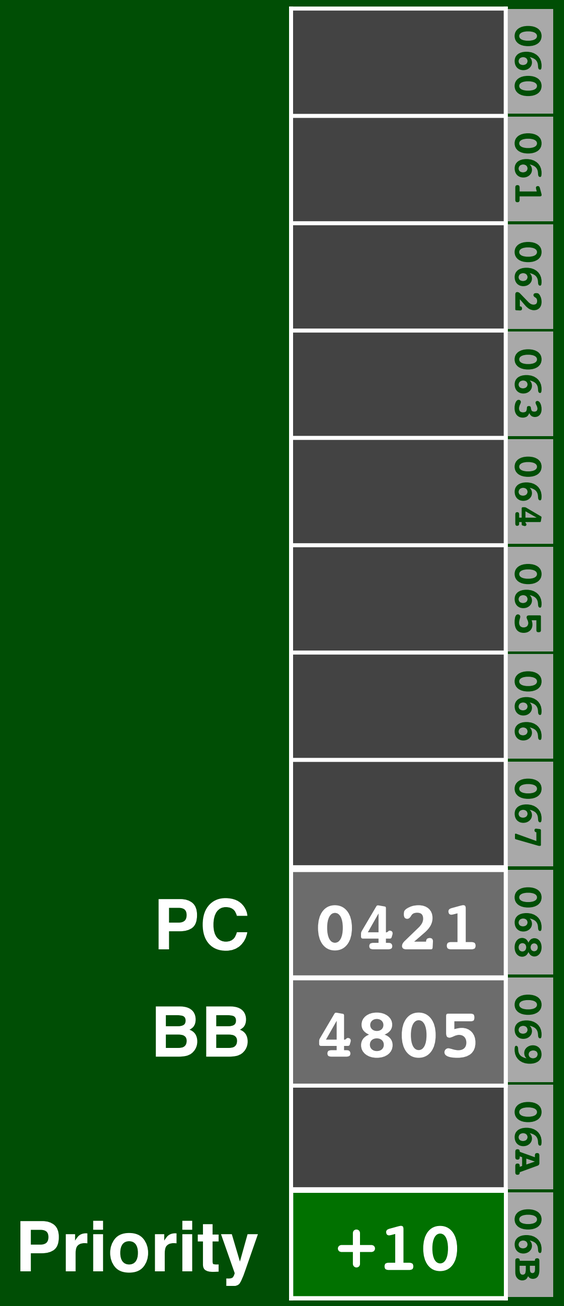
Job Entry

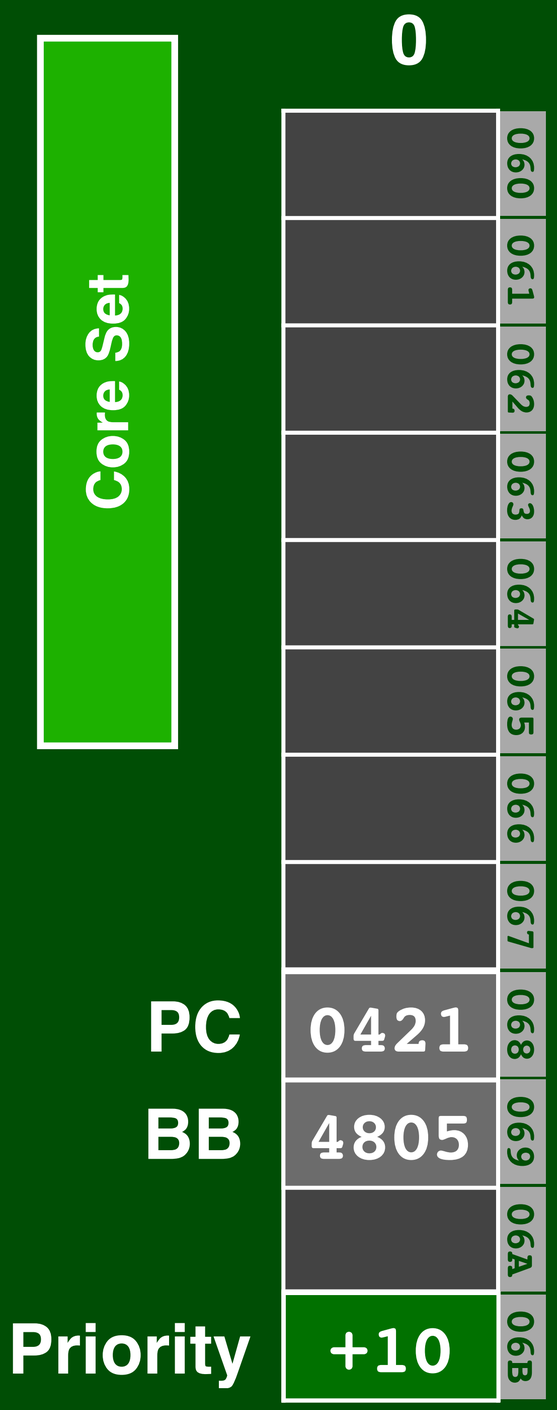

060
061
062
063
064
065
066
067
068
069
06A
06B
PC
0421
BB
4805
disabled
Priority
0 0 1 0 1 0
0-31

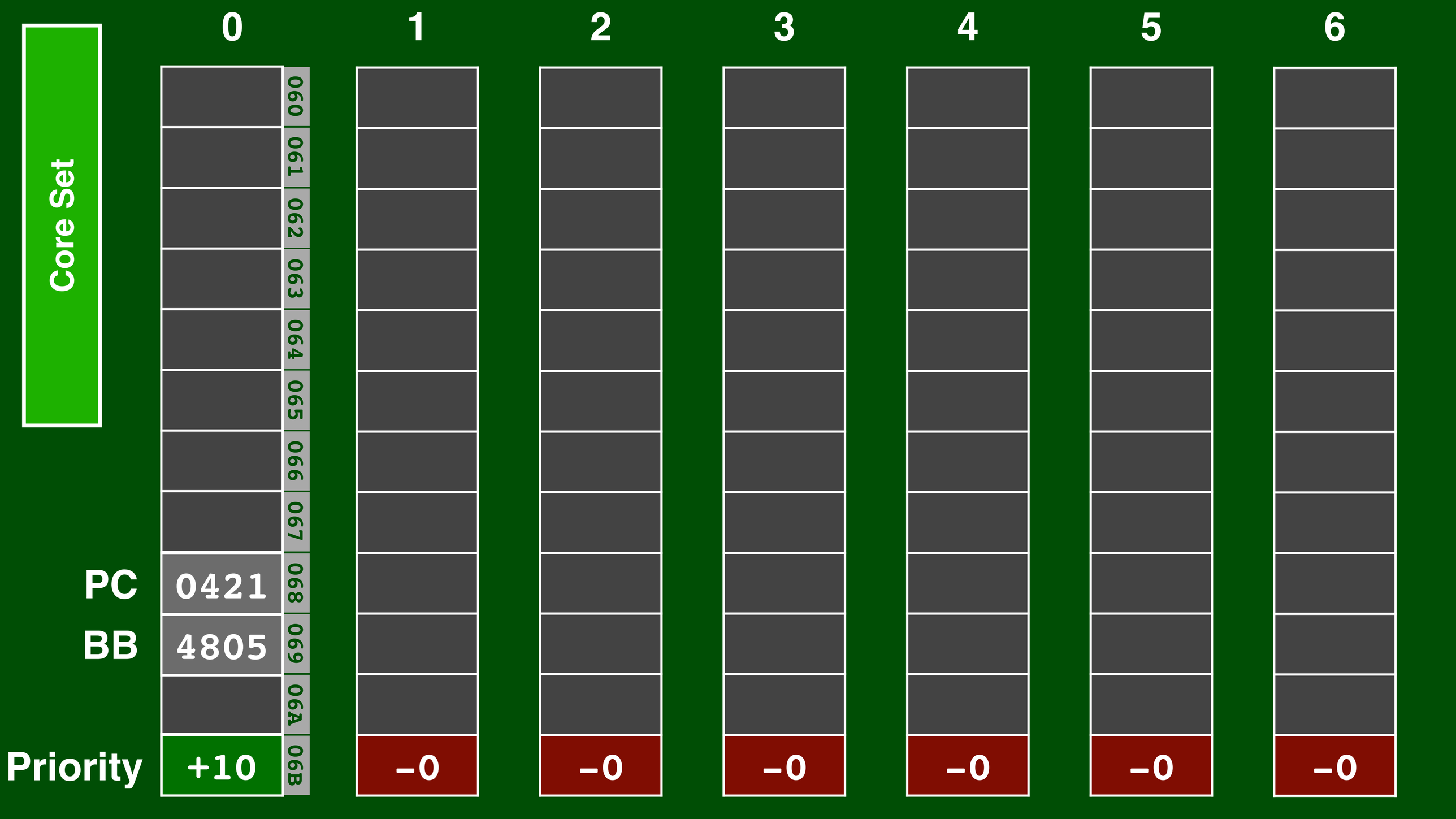
Job Entry

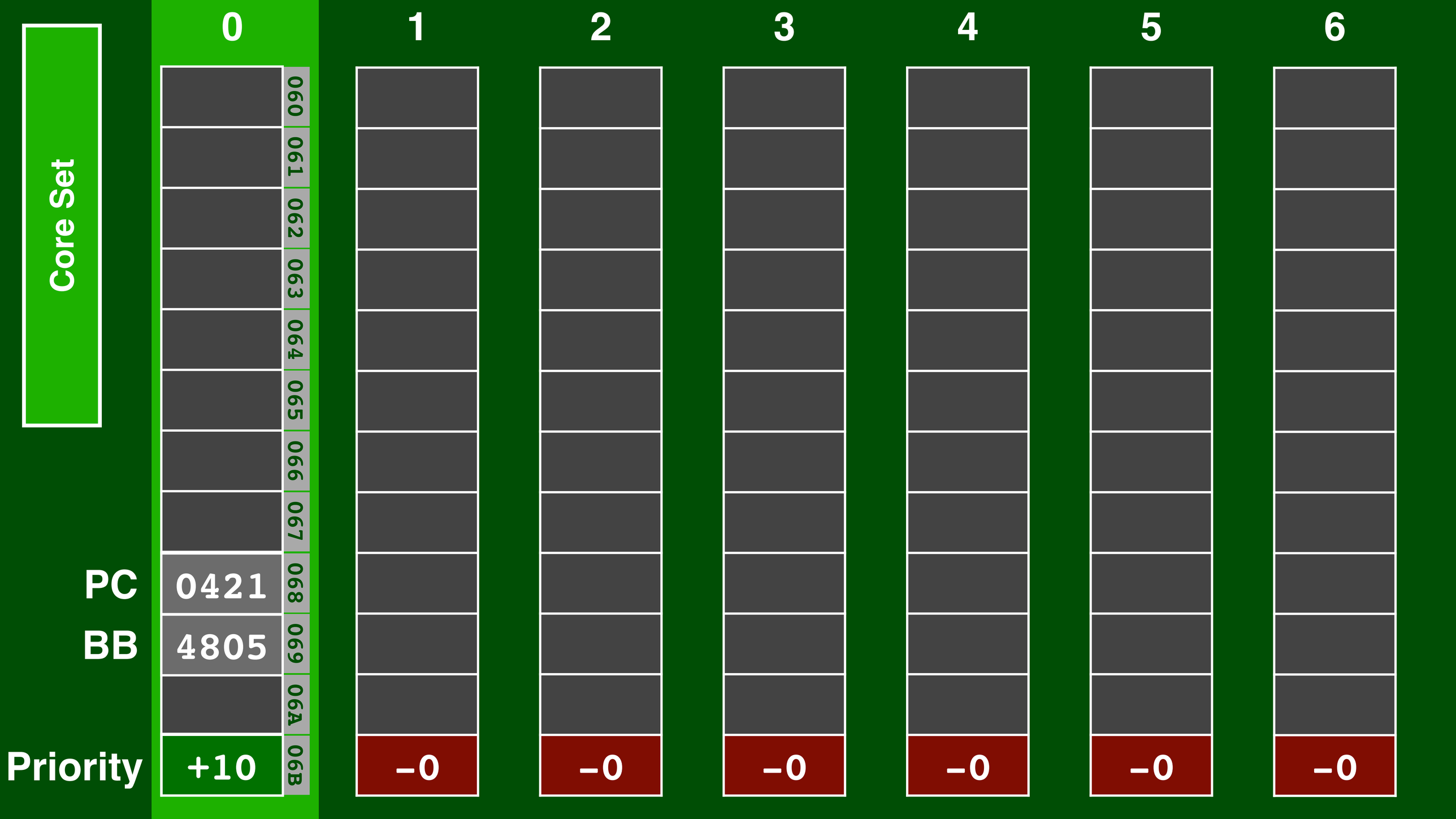

060
061
062
063
064
065
066
067
PC
0421
068
BB
4805
069
06A
Priority
1400
06B

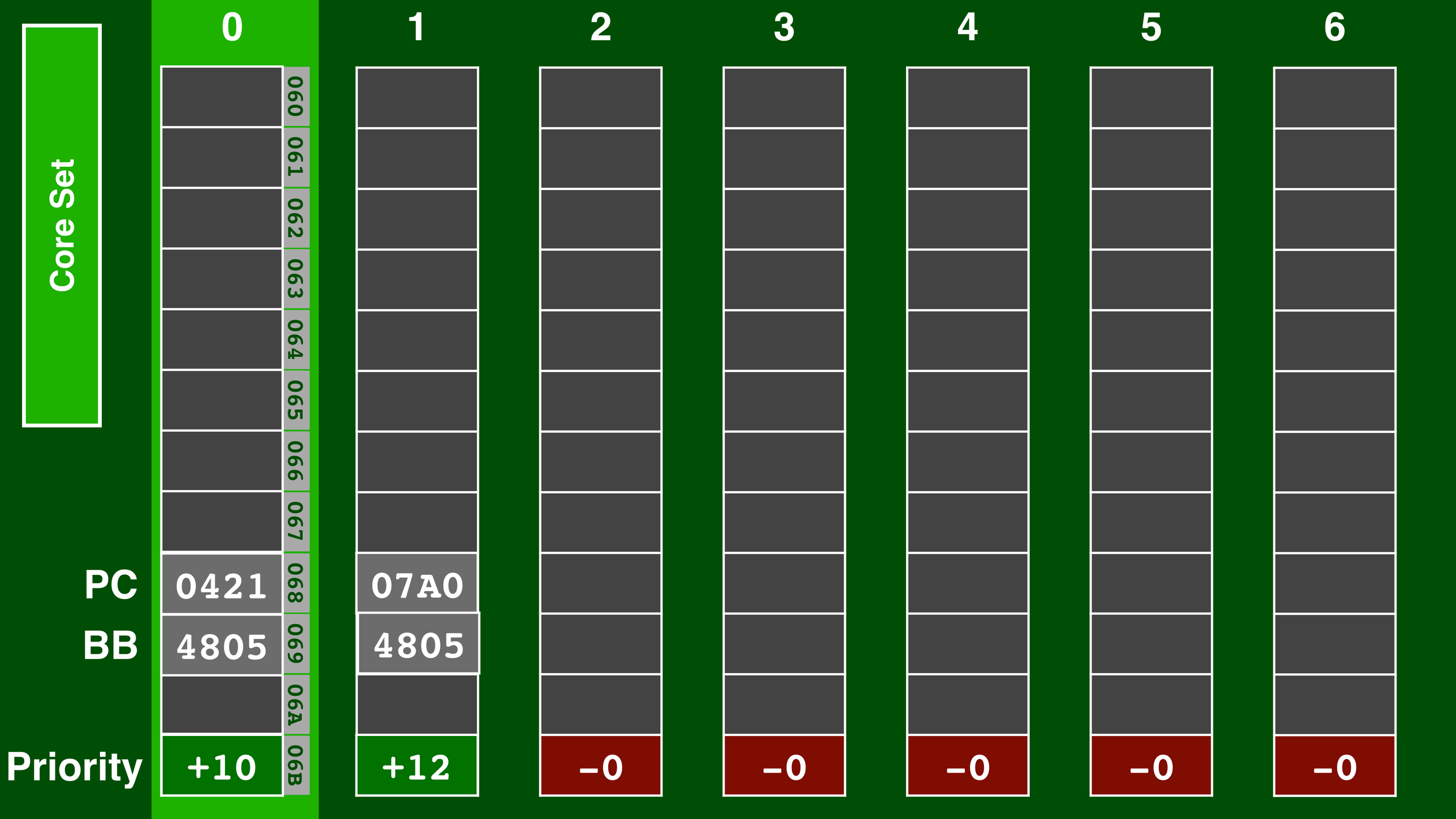
Job Entry

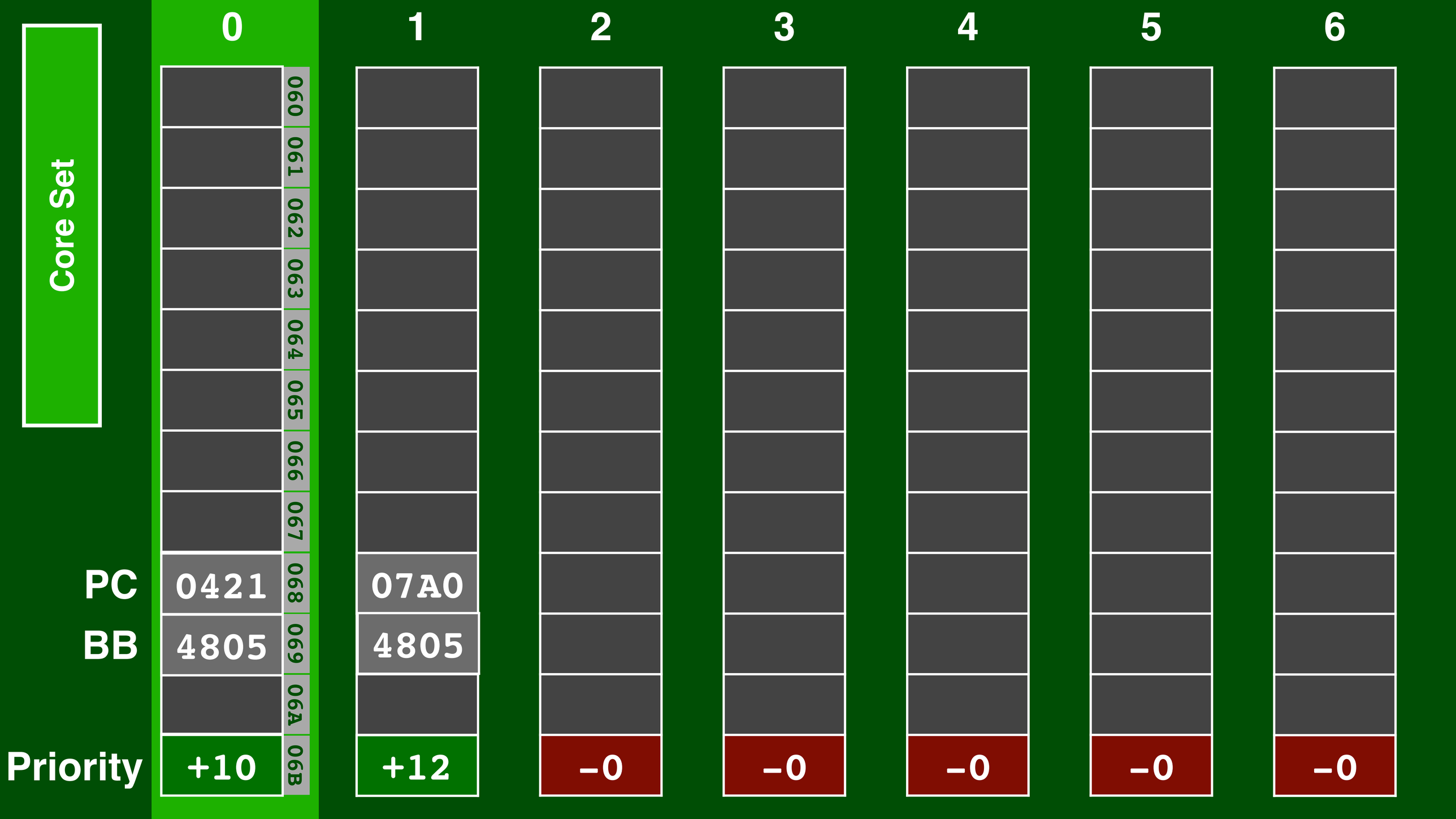

060
061
062
063
064
065
066
067
PC
0421
068
BB
4805
069
06A
Priority
+10
06B

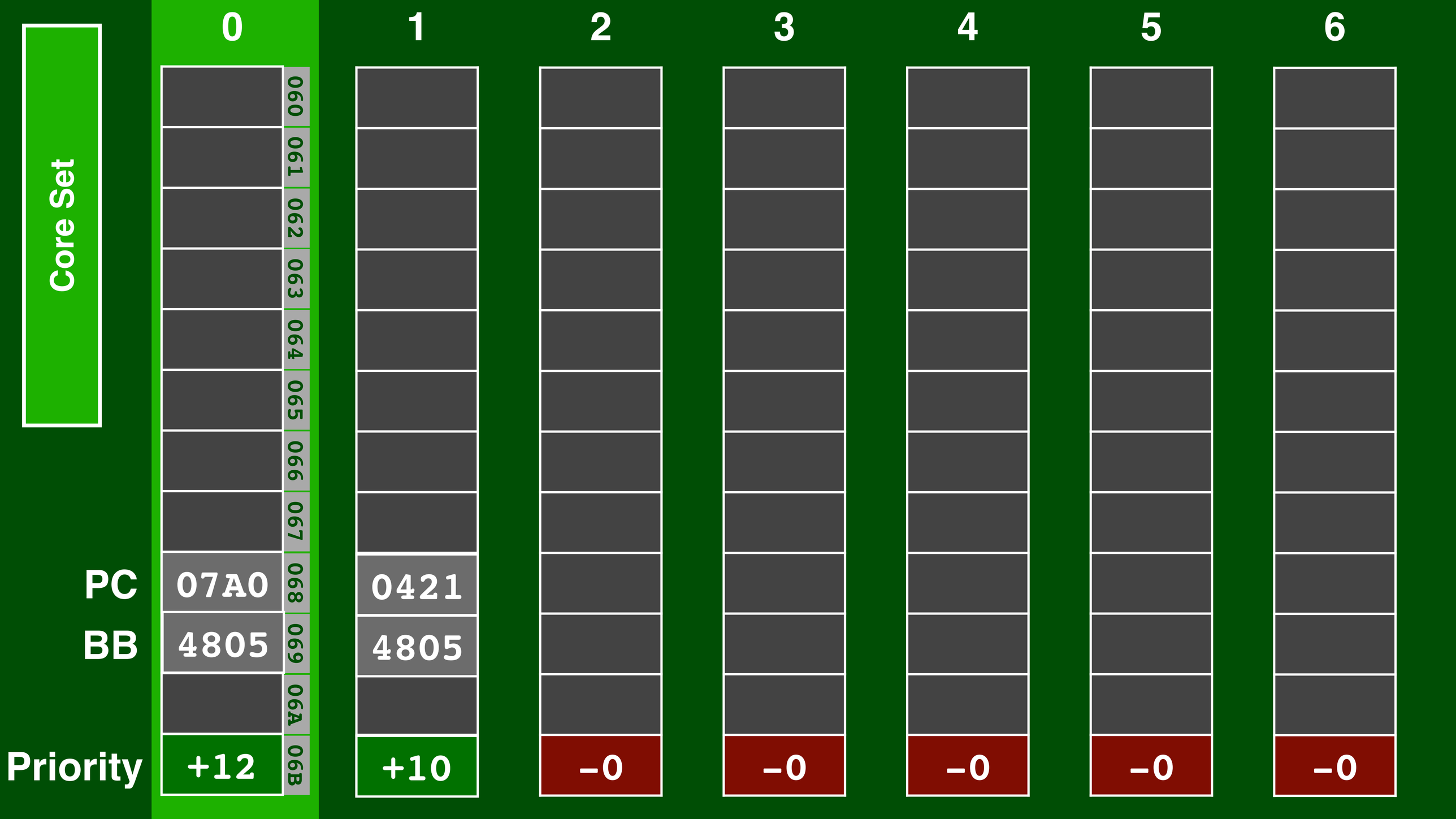

Core Set
0
060
061
062
063
064
065
066
067
PC
0421
068
BB
4805
069
06A
Priority
+10
06B

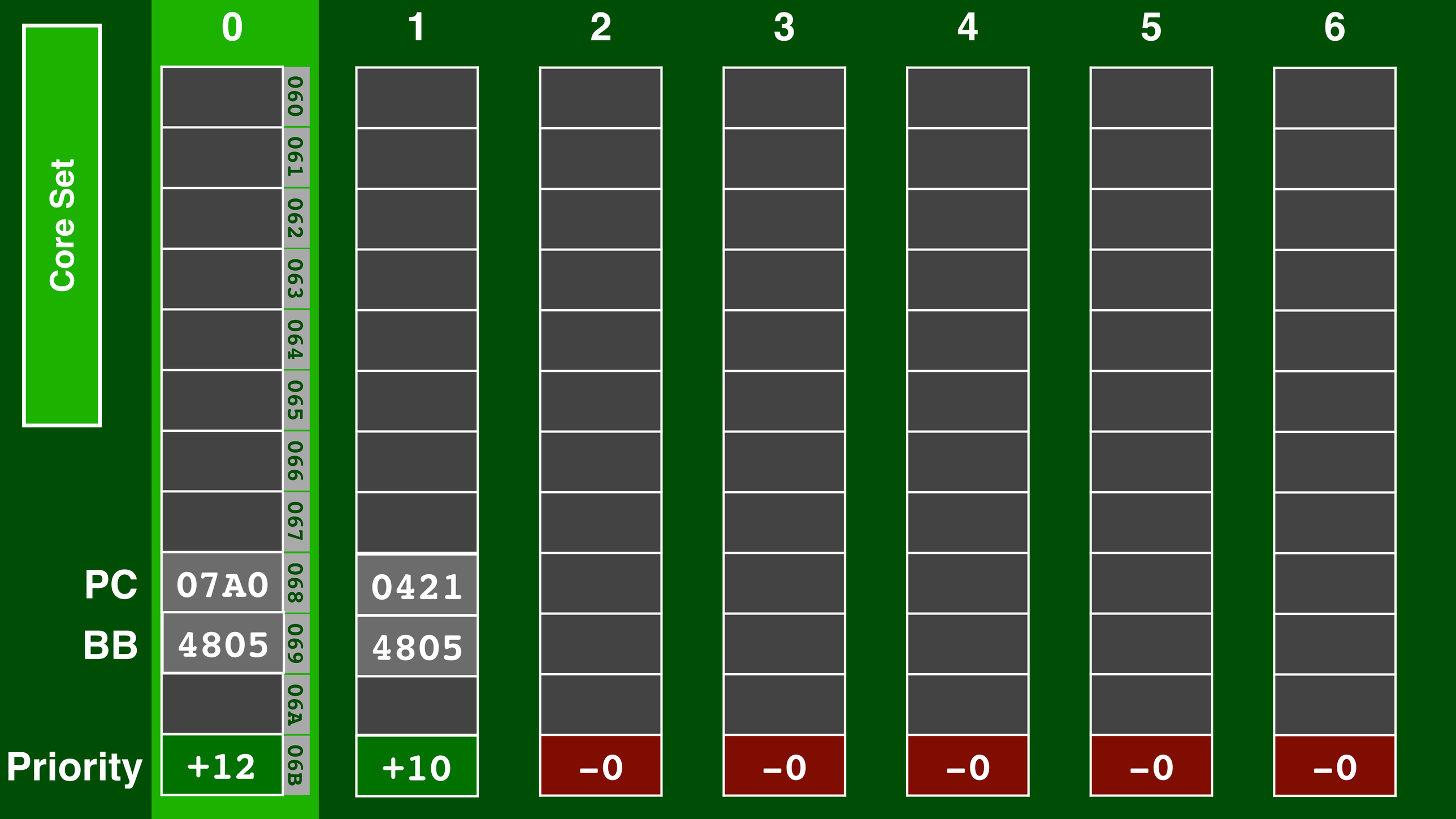

Core Set
0
1
2
3
4
5
6
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
0421
BB
4805
Priority
+10
-0
-0
-0
-0
-0
-0

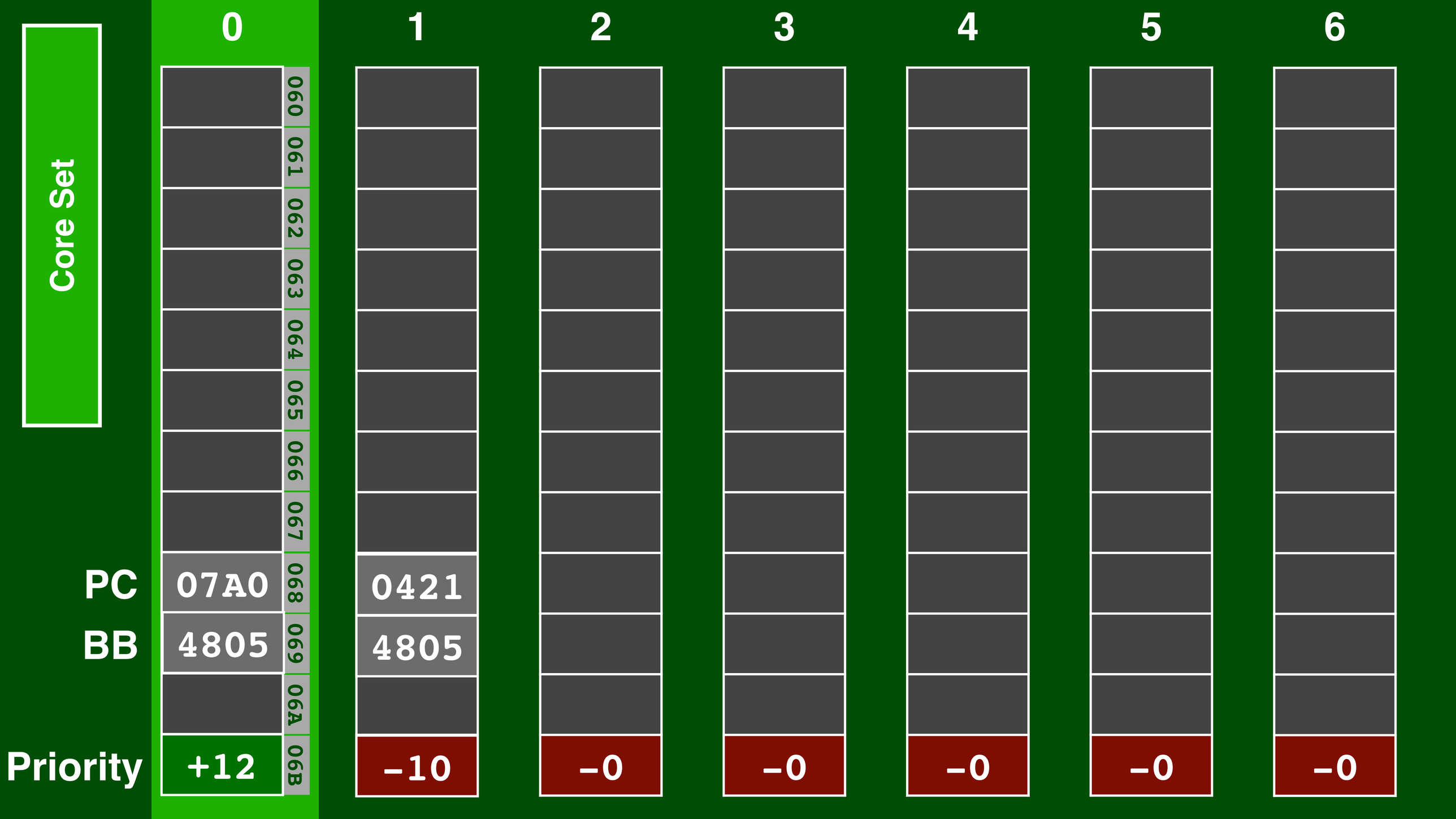

Core Set
0
1
2
3
4
5
6
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
0421
BB
4805
Priority
+10
-0
-0
-0
-0
-0
-0

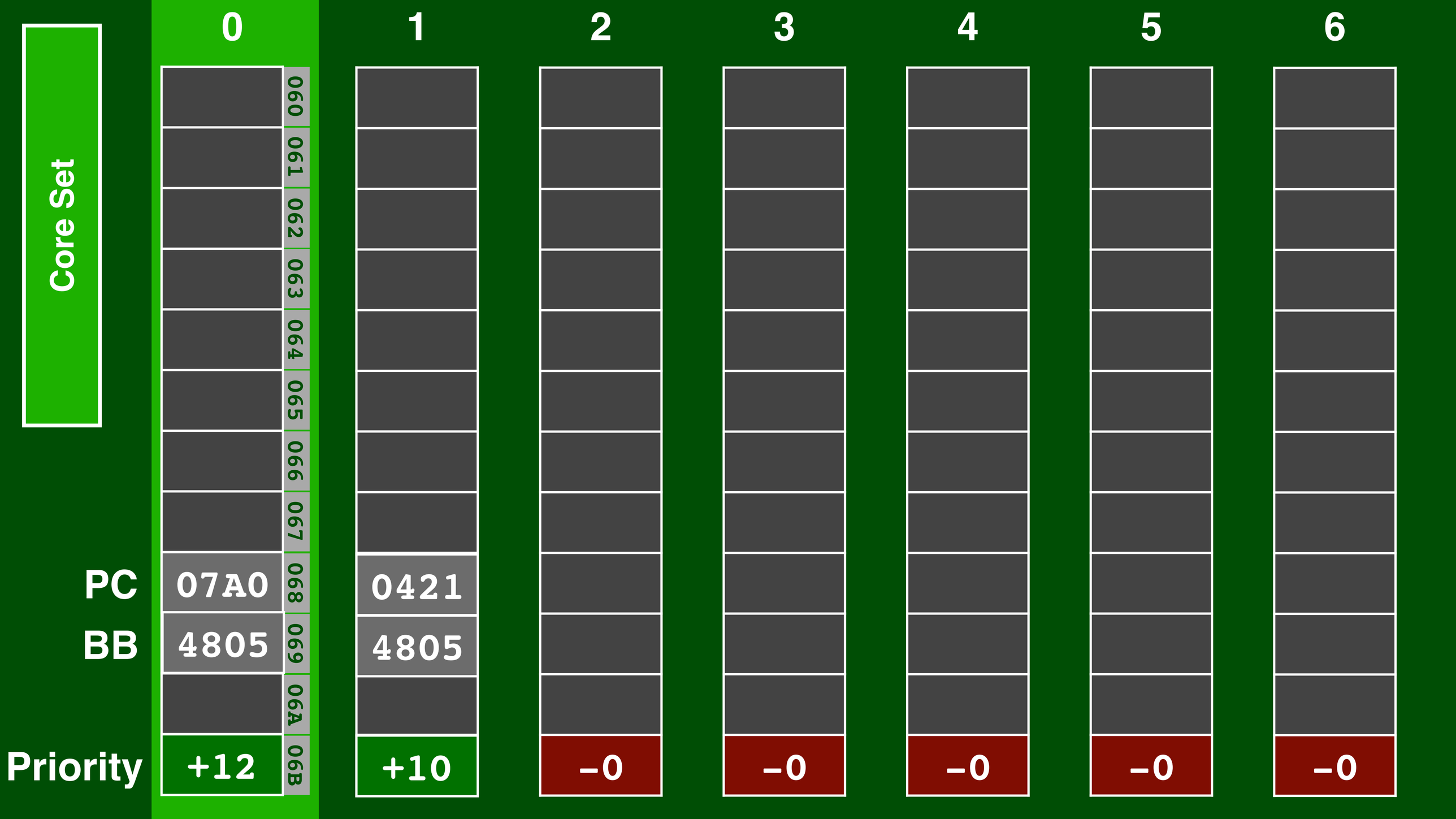

Core Set
0
1
2
3
4
5
6
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
0421
07A0
BB
4805
4805
Priority
+10
+12
-0
-0
-0
-0
-0

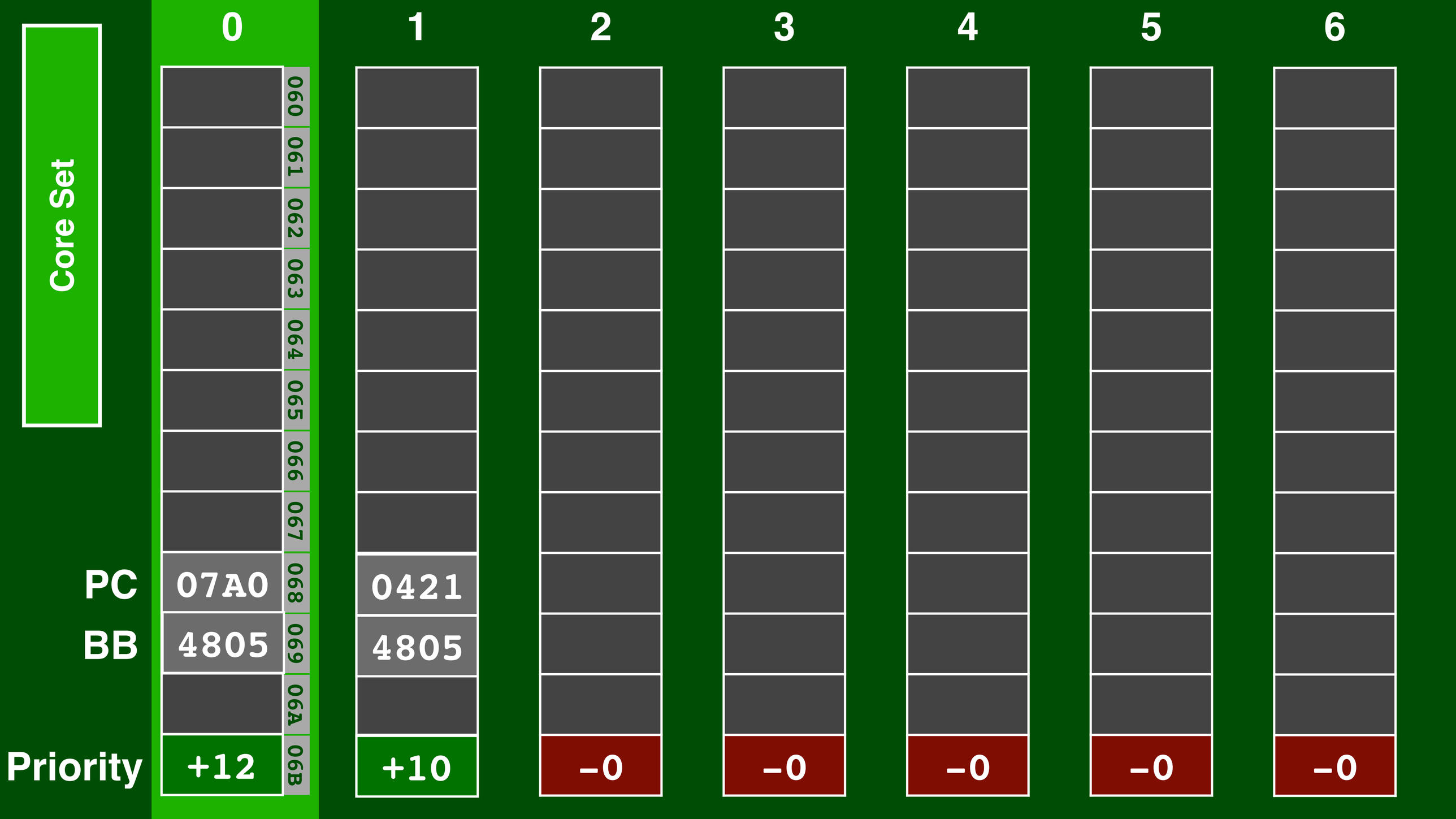

Core Set
0
1
2
3
4
5
6
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
0421
07A0
BB
4805
4805
Priority
+10
+12
-0
-0
-0
-0
-0

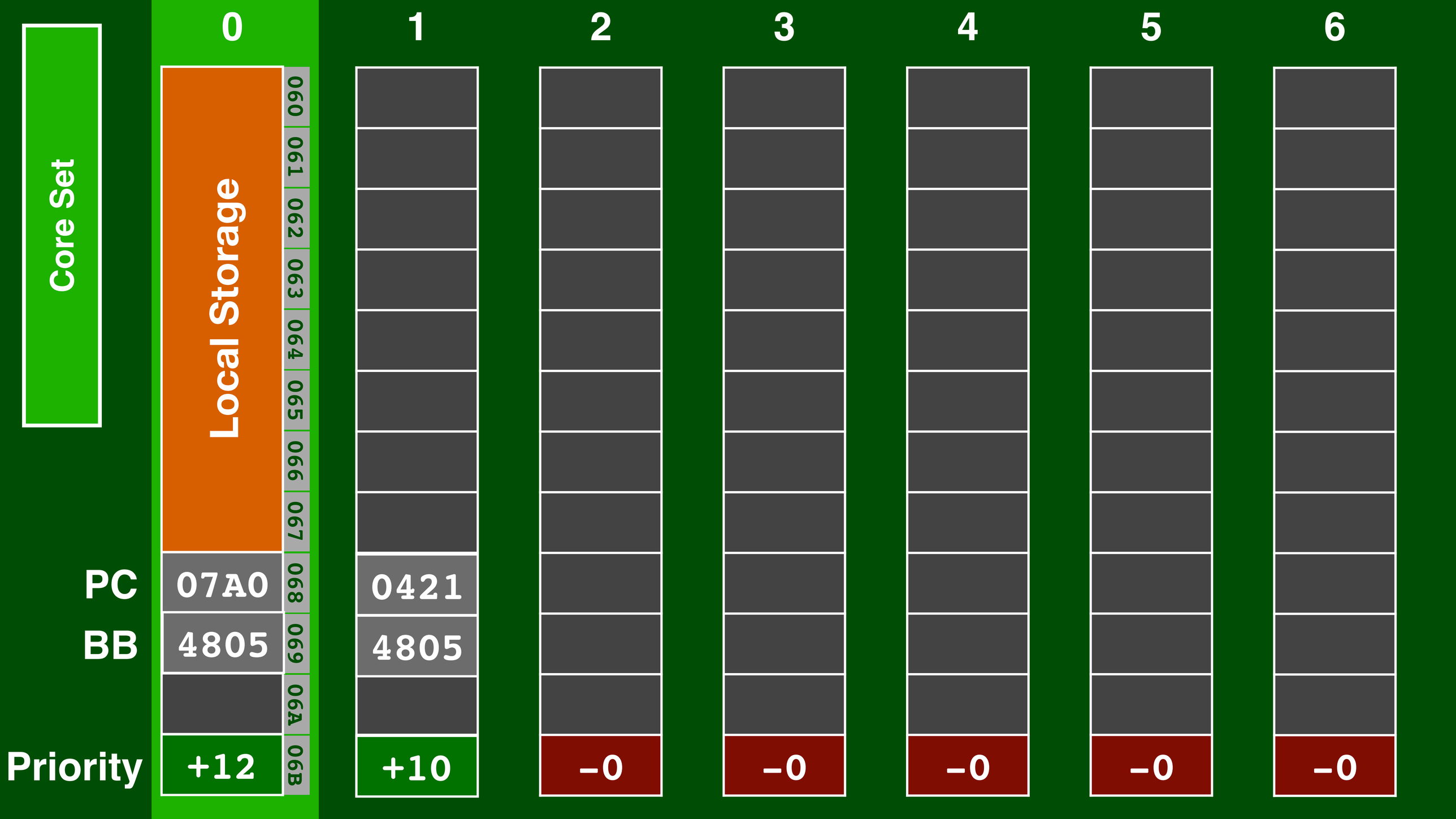

Core Set
0
1
2
3
4
5
6
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
BB
Priority
07A0
4805
+12
0421
4805
+10
-0
-0
-0
-0
-0

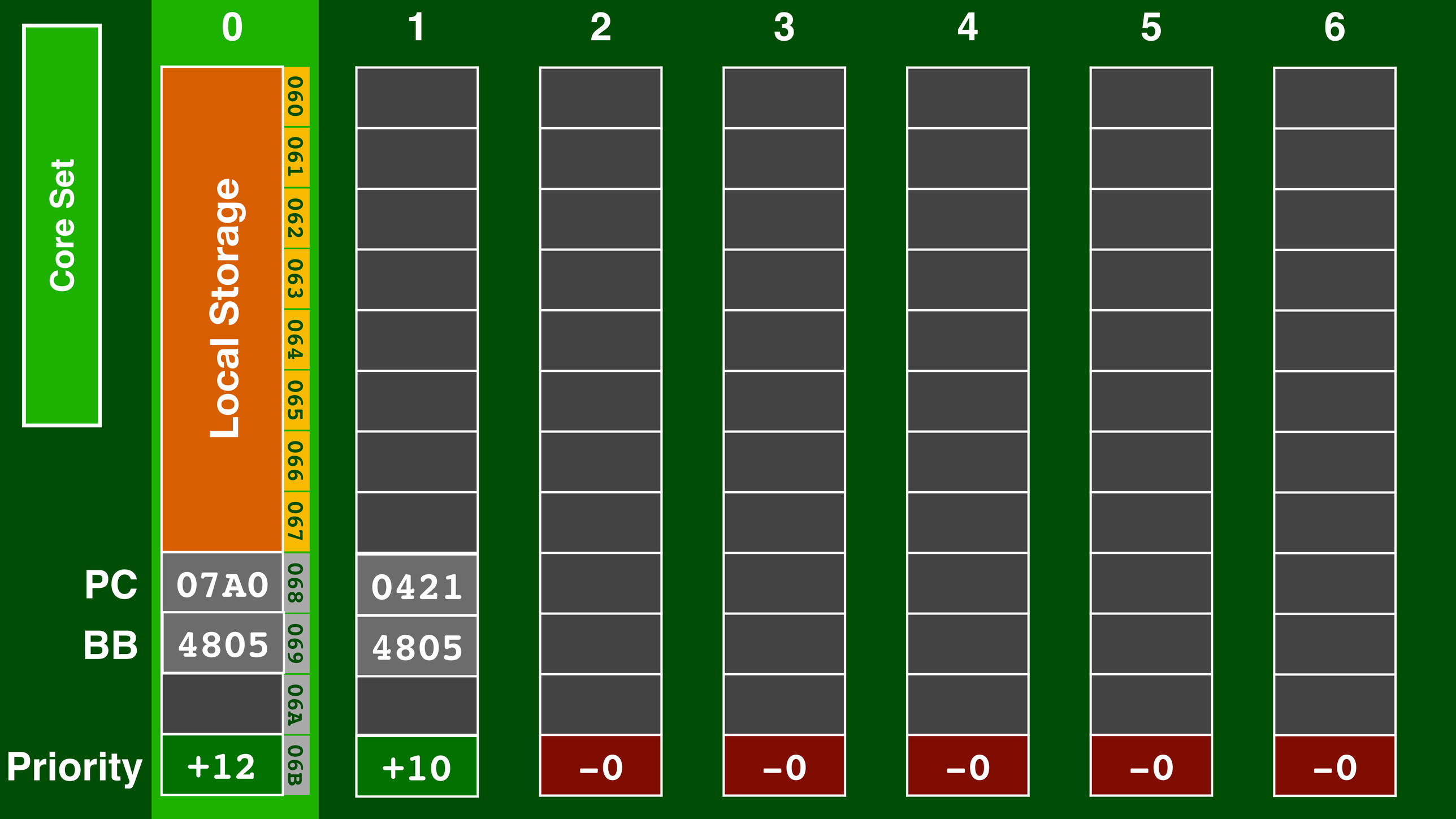

Core Set
0
1
2
3
4
5
6
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
BB
Priority
07A0
4805
+12
0421
4805
+10
-0
-0
-0
-0
-0

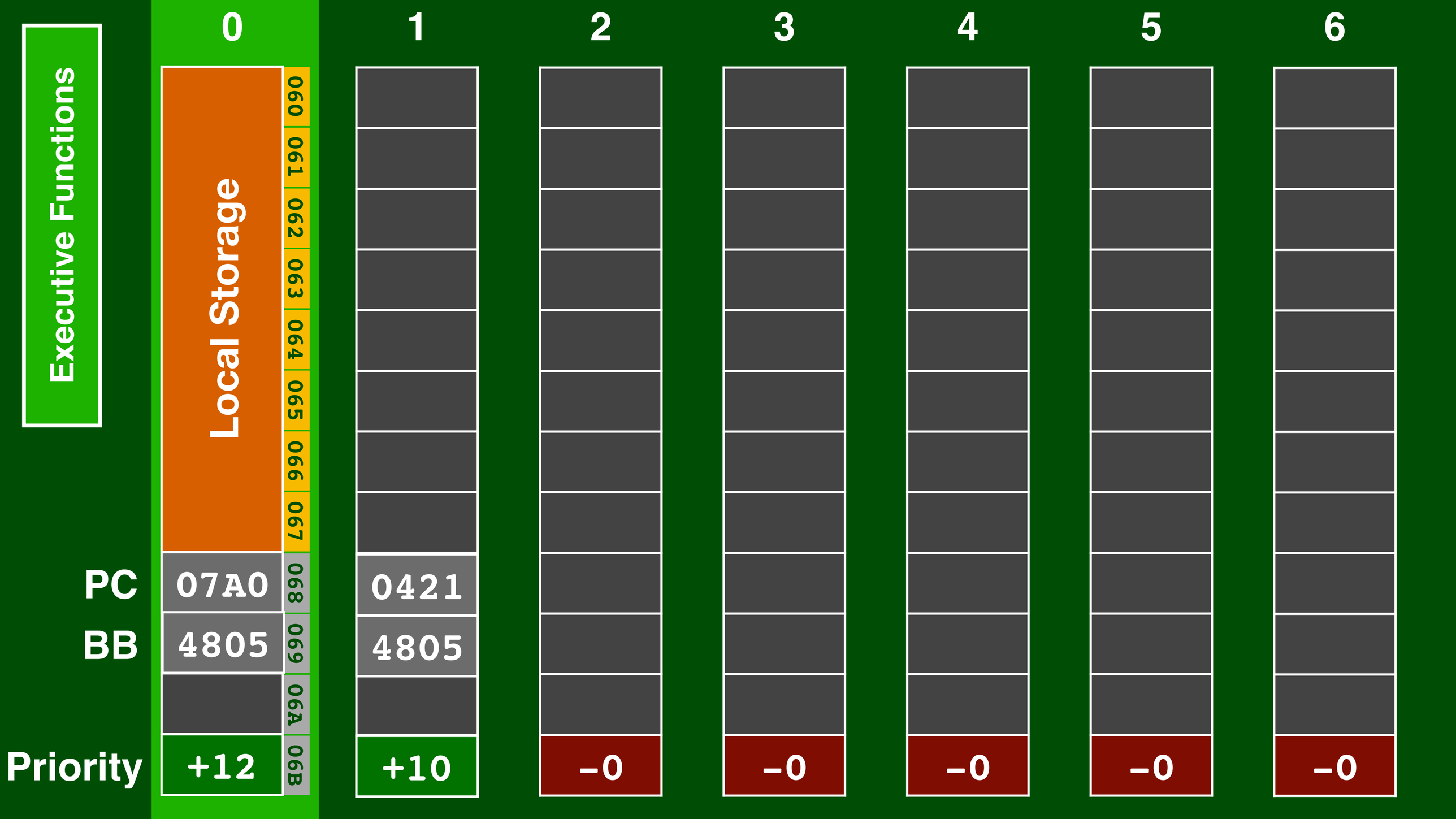

Core Set
0
1
2
3
4
5
6
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
BB
Priority
07A0
4805
+12
0421
4805
-10
-0
-0
-0
-0
-0

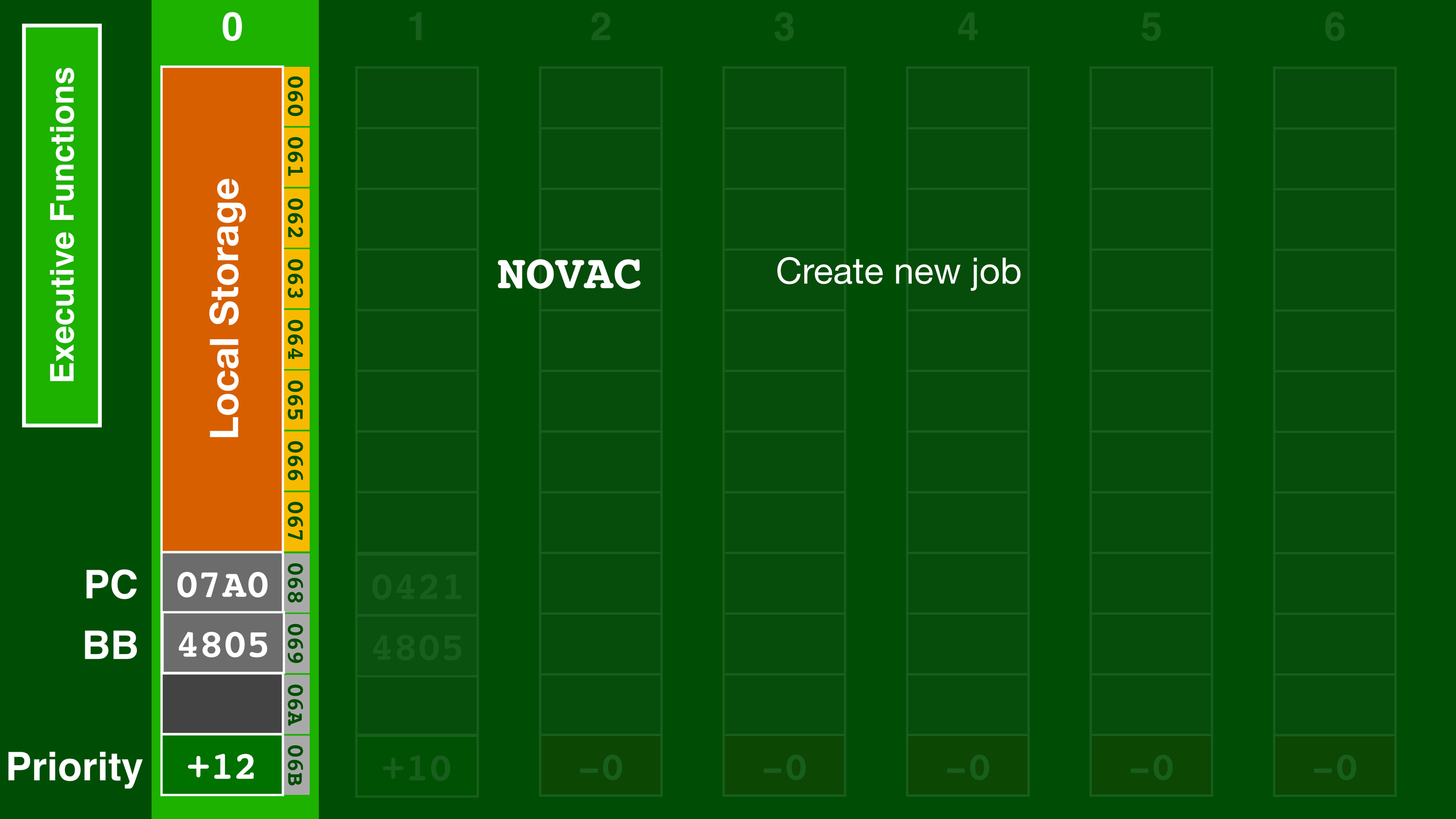

Core Set
0
1
2
3
4
5
6
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
BB
Priority
07A0
4805
+12
0421
4805
+10
-0
-0
-0
-0
-0

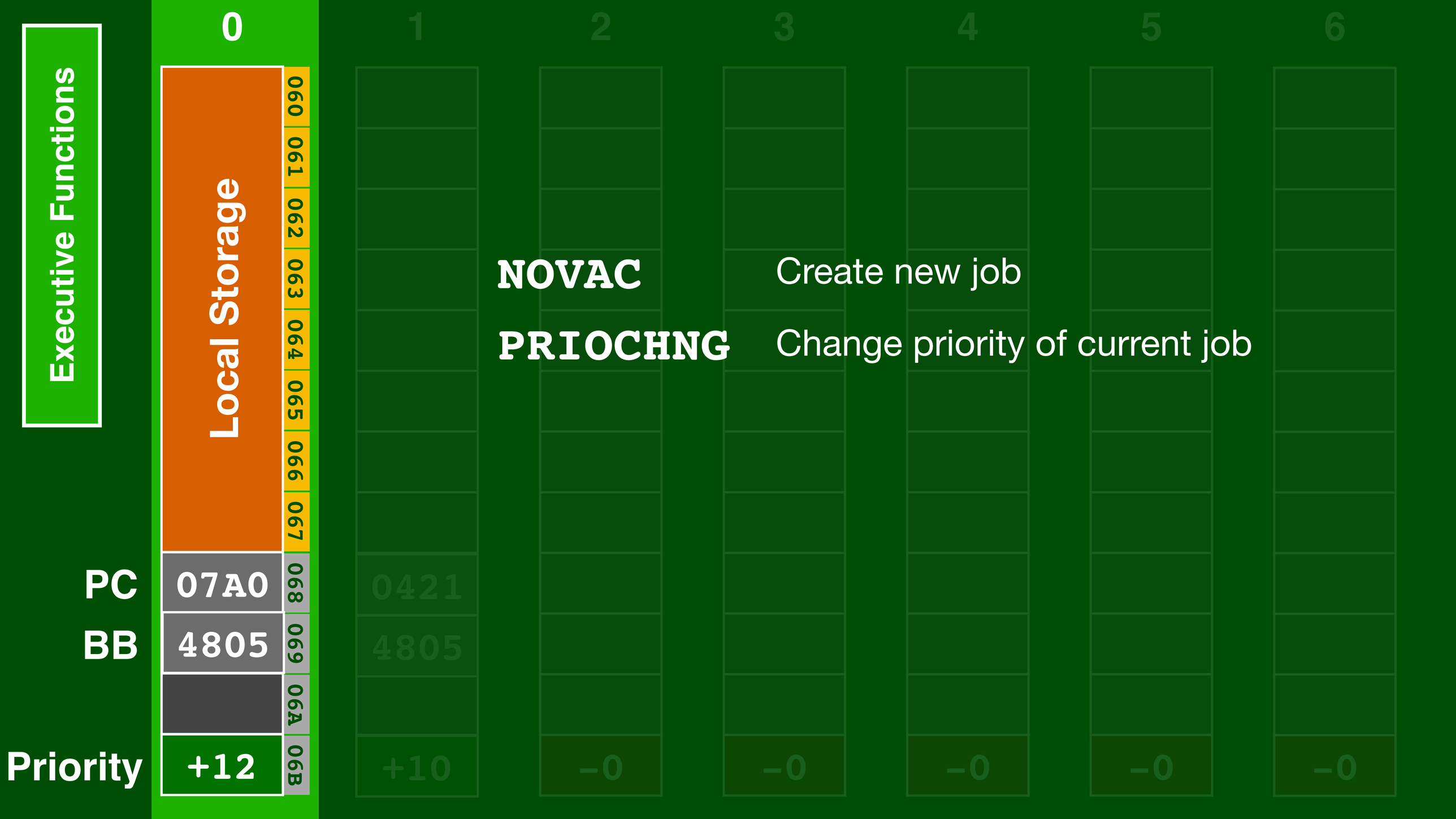

Core Set
0
1
2
3
4
5
6
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
BB
Priority
07A0
4805
+12
0421
4805
+10
-0
-0
-0
-0
-0

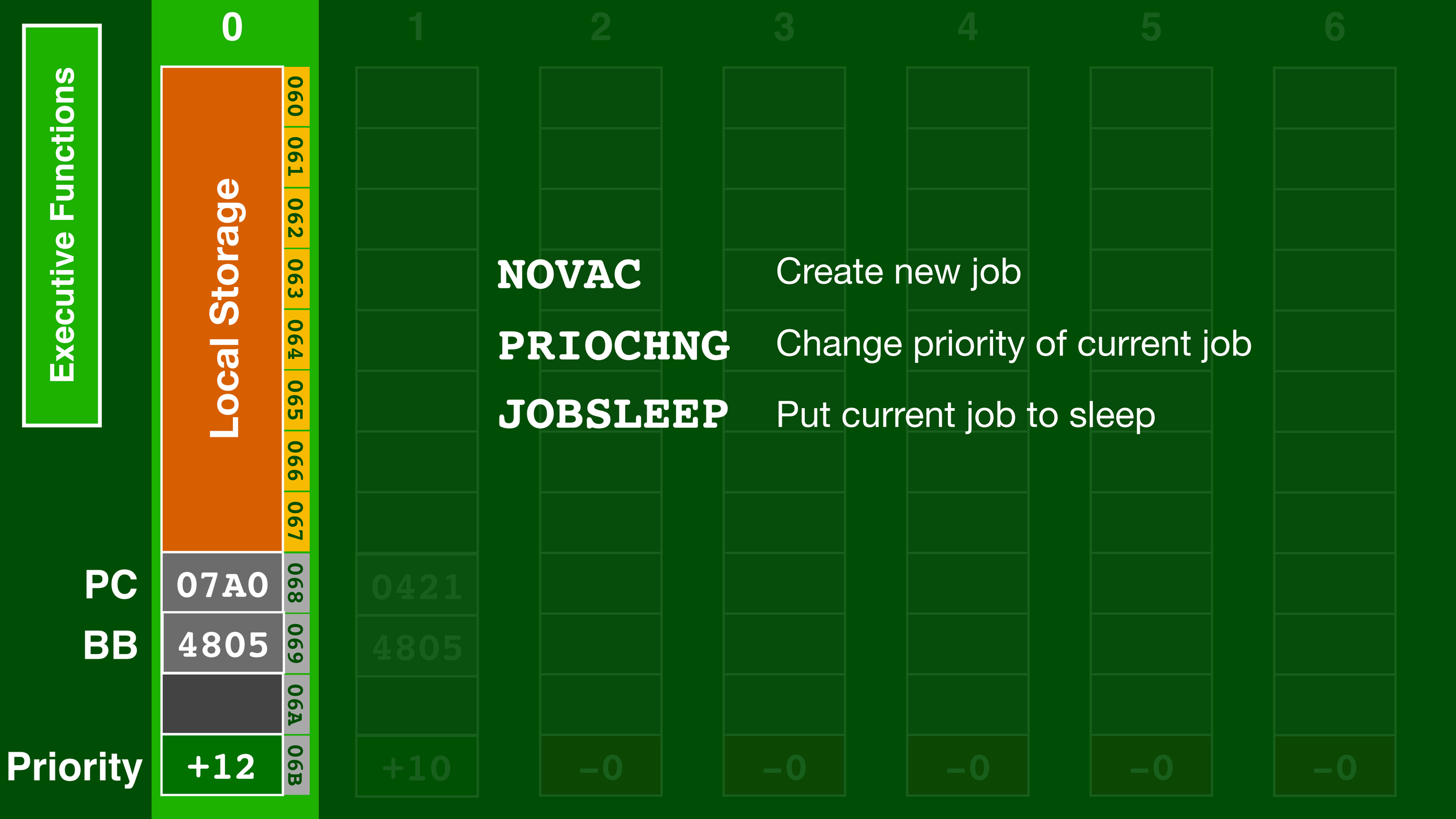

Core Set
0
1
2
3
4
5
6
Local Storage
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
BB
Priority
07A0
4805
+12
0421
4805
+10
-0
-0
-0
-0
-0


Core Set
0
1
2
3
4
5
6
Local Storage
060
061
062
063
064
065
066
067
PC
07A0
0421
068
BB
4805
4805
069
06A
Priority
+12
+10
-0
-0
-0
-0
-0
06B


Executive Functions
0
1
2
3
4
5
6
Local Storage
060
061
062
063
064
065
066
067
PC
07A0
0421
068
BB
4805
4805
069
06A
Priority
+12
+10
-0
-0
-0
-0
06B


Executive Functions
0
1
2
3
4
5
6
Local Storage
060
061
062
063
064
065
066
067
NOVAC
Create new job
PC
07A0
0421
BB
4805
4805
068
069
06A
06B
Priority
+12
+10
-0
-0
-0
-0


Executive Functions
0
1
2
3
4
5
6
Local Storage
060
061
062
063
064
065
066
067
PC
07A0
0421
068
BB
4805
4805
069
06A
Priority
+12
+10
-0
-0
-0
-0
06B
NOVAC
Create new job
PRIOCHNG
Change priority of current job

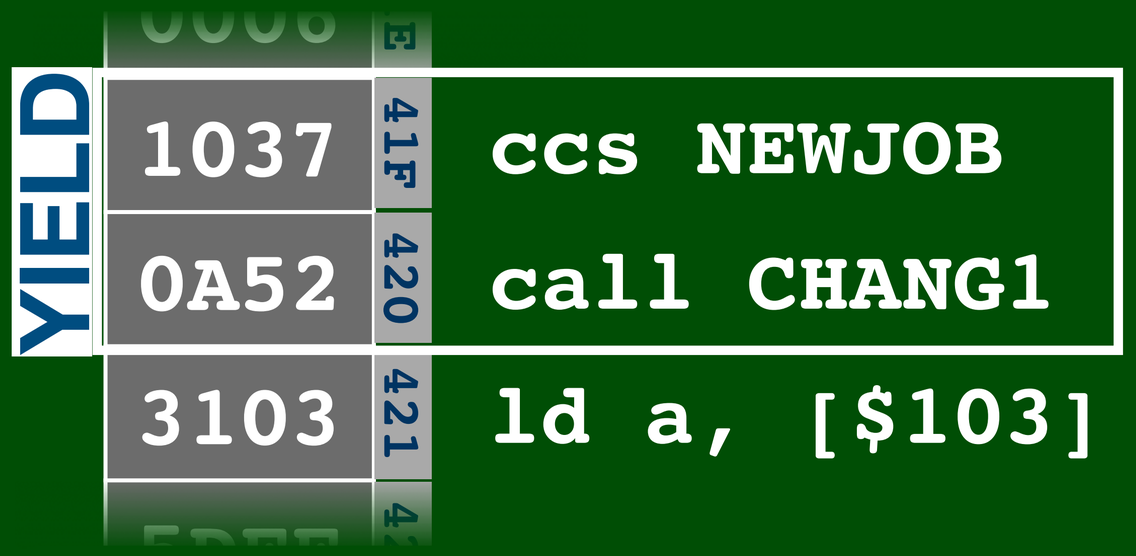

Executive Functions
0
1
2
3
4
5
6
Local Storage
060
061
062
063
064
065
066
067
PC
07A0
0421
068
BB
4805
4805
069
06A
Priority
+12
+10
-0
-0
-0
-0
06B
NOVAC
Create new job
PRIOCHNG
Change priority of current job
JOBSLEEP
Put current job to sleep


Executive Functions
0
1
2
3
4
5
6
Local Storage
060
061
062
063
064
065
066
067
PC
07A0
0421
068
BB
4805
4805
069
06A
Priority
+12
+10
-0
-0
-0
-0
06B
NOVAC
Create new job
PRIOCHNG
Change priority of current job
JOBSLEEP
Put current job to sleep
JOBWAKE
Wake a job

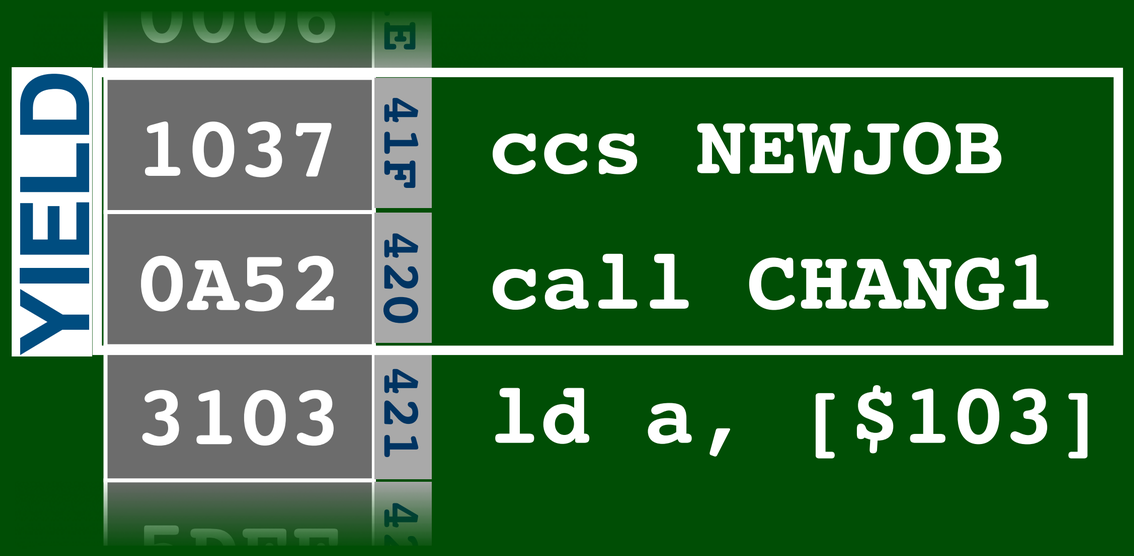

Executive Functions
0
1
2
3
4
5
6
Local Storage
060
061
062
063
064
065
066
067
068
069
06A
06B
PC
BB
Priority
07A0
4805
+12
0421
4805
+10
-0
-0
-0
-0
NOVAC
Create new job
PRIOCHNG
Change priority of current job
JOBSLEEP
Put current job to sleep
JOBWAKE
Wake a job
ENDOFJOB
End current job

Yield

Yield


YIELD
1037
41F
ccs NEWJOB
0A52
420
call CHANG1
3103
421
ld a, [$103]

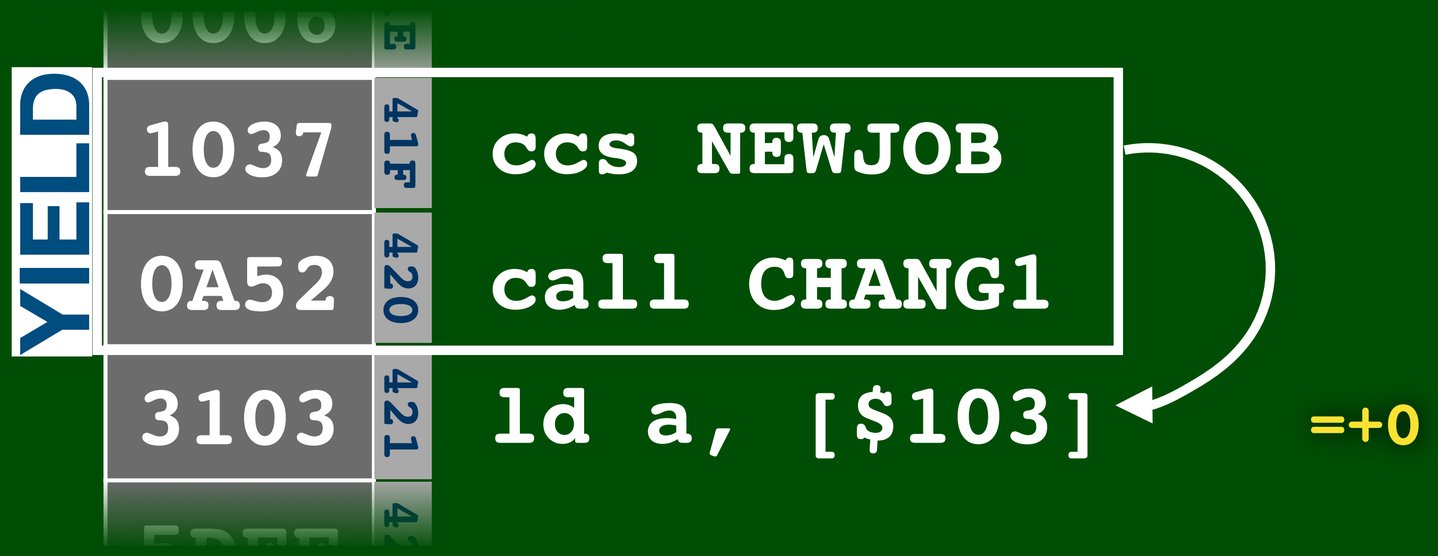
Yield



NEWJOB 0000 037

Job with Highest Priority

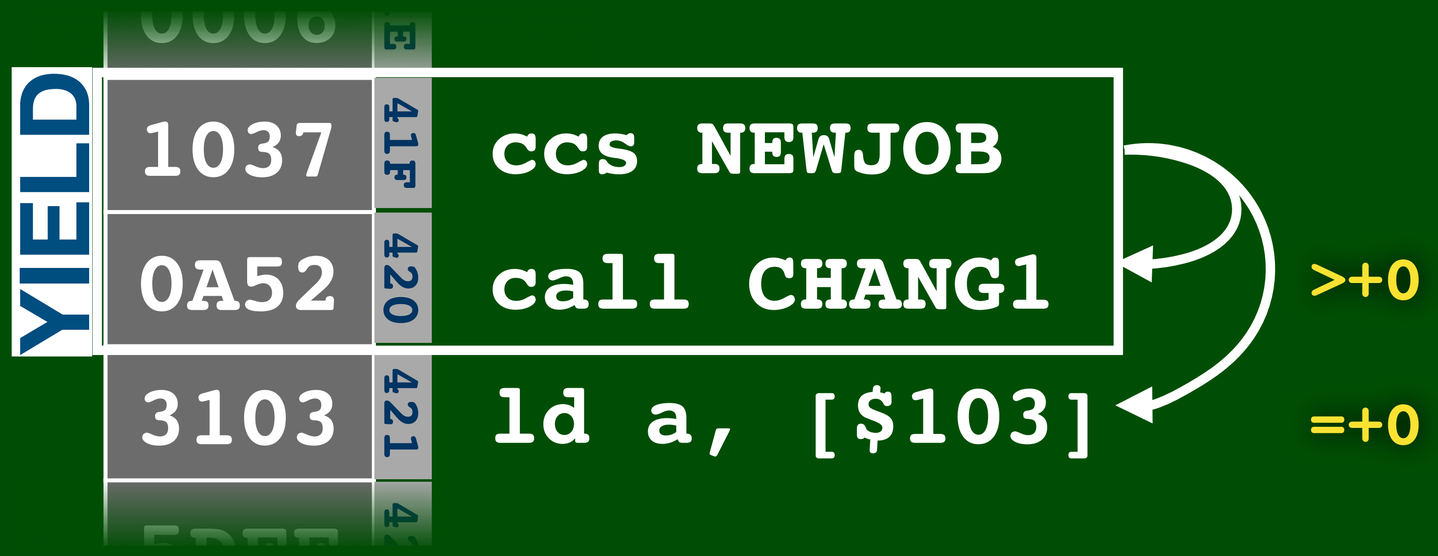
Yield



NEWJOB 0000 037

Job with Highest Priority

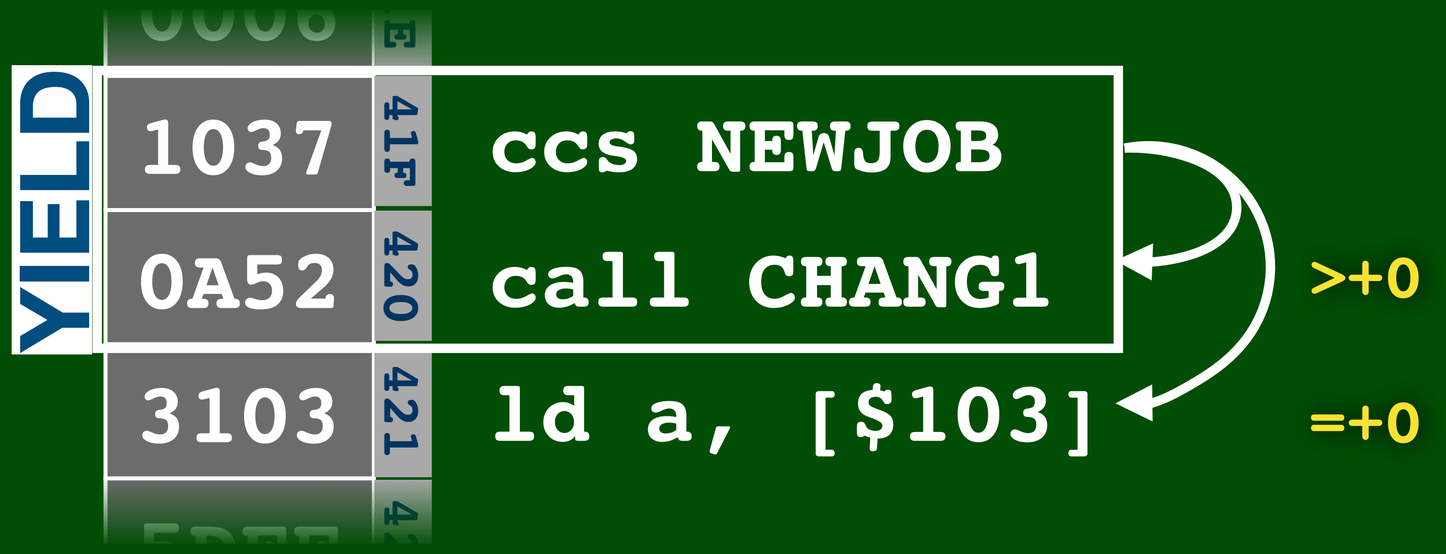
Yield



NEWJOB 0000 037

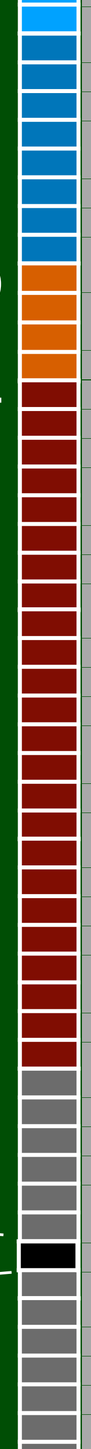
Job with Highest Priority

Yield



NEWJOB 0000 037

Job with Highest Priority



Math Support

Math Support

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

Math Support

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

Single

.1234

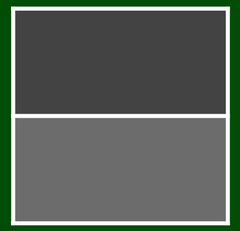
Math Support

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

Single

.1234

Double



.12345678

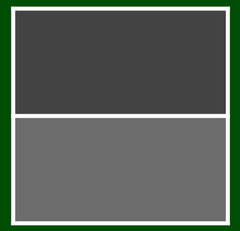
Math Support

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

Single

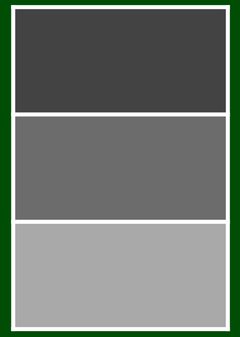
.1234

Double



.12345678

Triple



.12345678012

Math Support

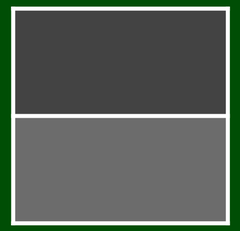
$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

Single

.1234

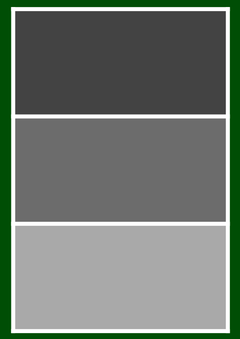
Vector

Double



.12345678

Triple



.12345678012

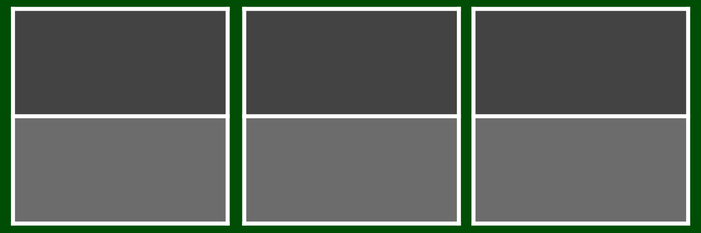
Math Support

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

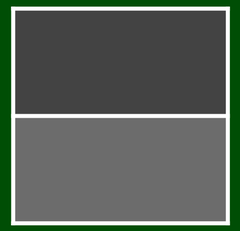
Single

.1234

Vector

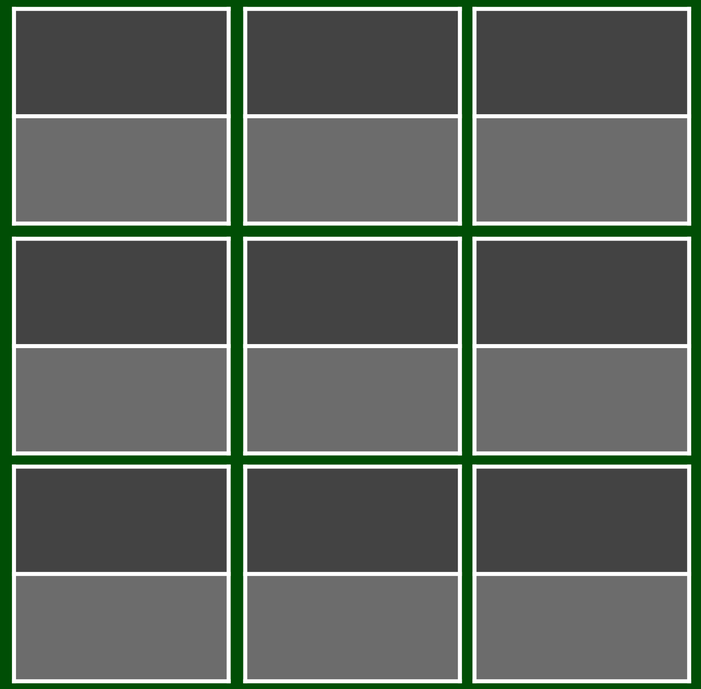


Double

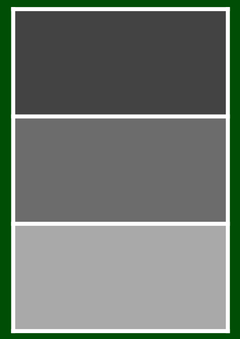


.12345678

Matrix



Triple



.12345678012

Math Support

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

Single

.1234

Vector

-1

Double

+0

.12345678

Matrix

Triple

+1

.12345678012

MPAC

MODE

Math Support

Single

.1234

Double

+0

.12345678

Triple

+1

.12345678012

Vector

-1

Matrix

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

```
load_vector(vector_t *v1);
mul_double(double_t *a);
push();
load_vector(vector_t *v2);
mul_double(double_t *b);
add_double(pop());
```

MPAC

MODE

Math Support

Single

.1234

Vector

-1

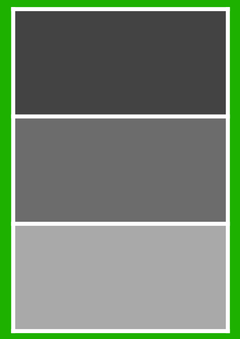
Double

+0

.12345678

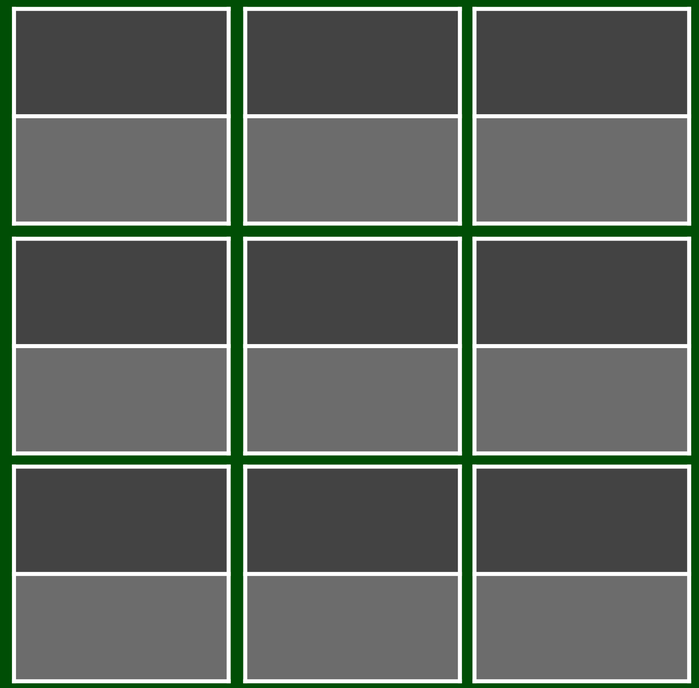
Triple

+1



.12345678012

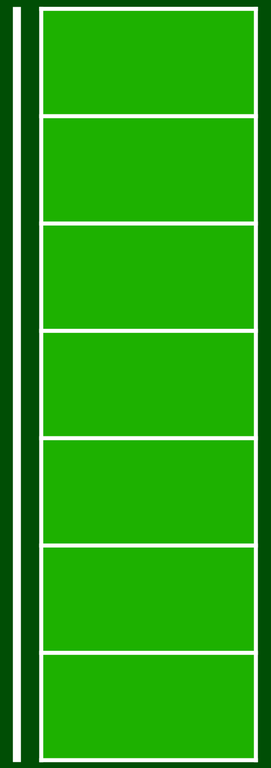
Matrix



$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

```
load_vector(vector_t *v1);
mul_double(double_t *a);
push();
load_vector(vector_t *v2);
mul_double(double_t *b);
add_double(pop());
```

MPAC



MODE

Math Support

Single

.1234

Vector

-1

Double

+0

.12345678

Triple

+1

.12345678012

Matrix

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

```
load_vector(vector_t *v1);
mul_double(double_t *a);
push();
load_vector(vector_t *v2);
mul_double(double_t *b);
add_double(pop());
```

MPAC

MODE

Math Support

Single

.1234

Double

+0

.12345678

Triple

+1

.12345678012

Vector

-1

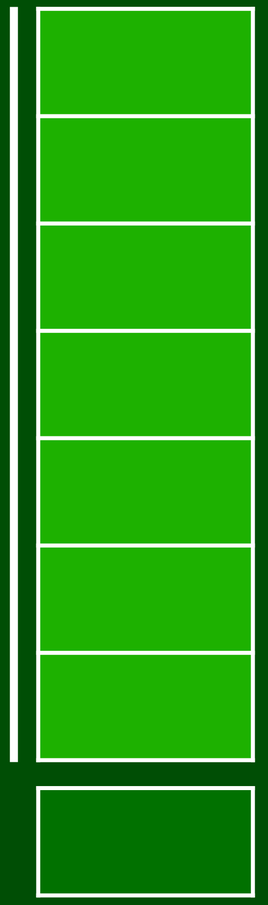
Matrix

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

```
load_vector(vector_t *v1);
mul_double(double_t *a);
push();
load_vector(vector_t *v2);
mul_double(double_t *b);
add_double(pop());
```

MPAC

MODE



Math Support

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

```
load_vector(vector_t *v1);
mul_double(double_t *a);
push();
load_vector(vector_t *v2);
mul_double(double_t *b);
add_double(pop());
```

Single

.1234

Double

+0

.12345678

Triple

+1

.12345678012

Vector

-1

Matrix

MPAC

MODE

Math Support

Single

.1234

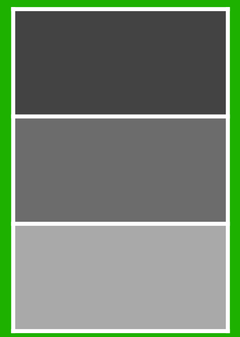
Double

+0

.12345678

Triple

+1

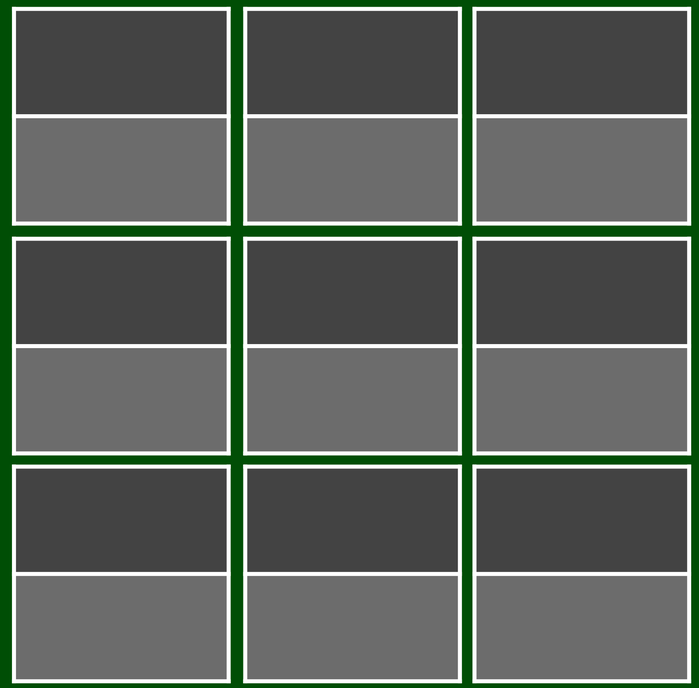


.12345678012

Vector

-1

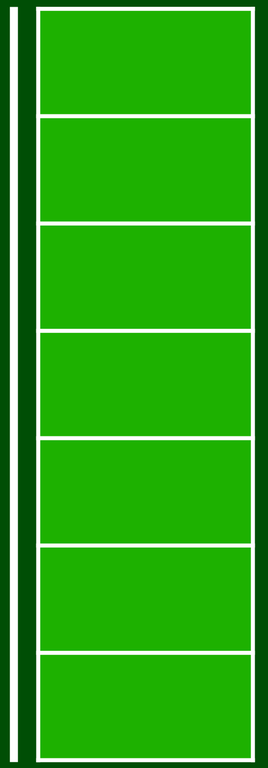
Matrix



$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

```
load_vector(vector_t *v1);
mul_double(double_t *a);
push();
load_vector(vector_t *v2);
mul_double(double_t *b);
add_double(pop());
```

MPAC



MODE

Math Support

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

```
load_vector(vector_t *v1);
mul_double(double_t *a);
push();
load_vector(vector_t *v2);
mul_double(double_t *b);
add_double(pop());
```

Single

.1234

Vector

-1

Double

+0

.12345678

Matrix

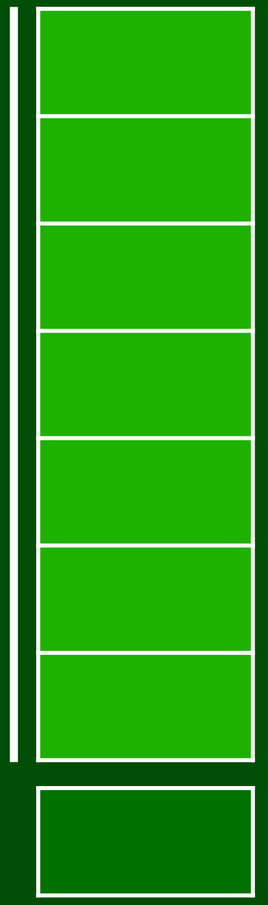
Triple

+1

.12345678012

MPAC

MODE



Math Support

Single

.1234

Double

+0

.12345678

Triple

+1

.12345678012

Vector

-1

Matrix

$$a \cdot \vec{v_1} + b \cdot \vec{v_2}$$

```
load_vector(vector_t *v1);
mul_double(double_t *a);
push();
load_vector(vector_t *v2);
mul_double(double_t *b);
add_double(pop());
```

MPAC

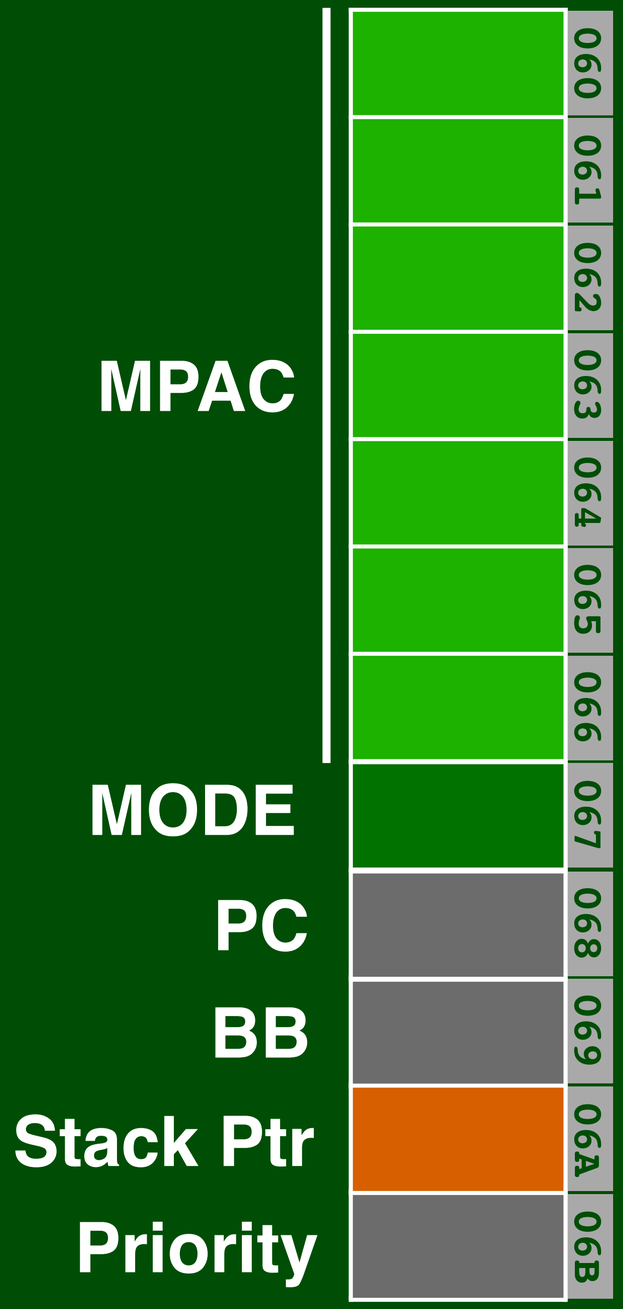
MODE

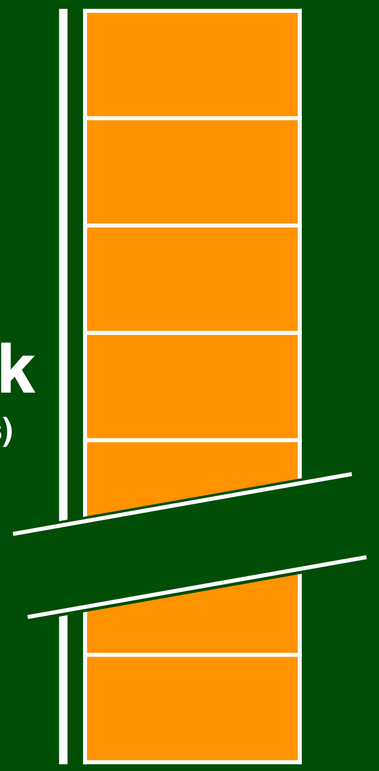
Stack
(38 words)

Stack Ptr

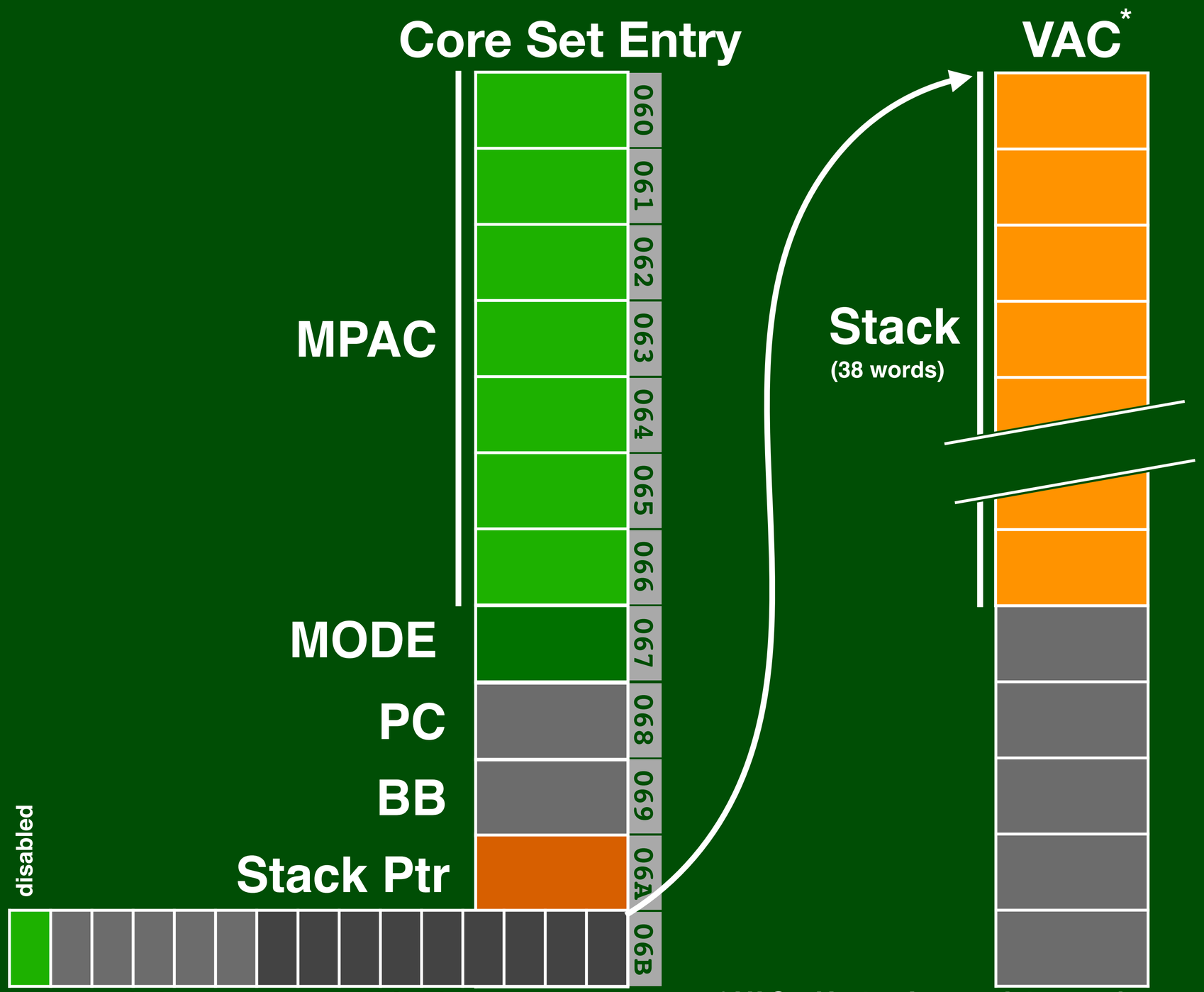
VAC

Core Set Entry





VAC

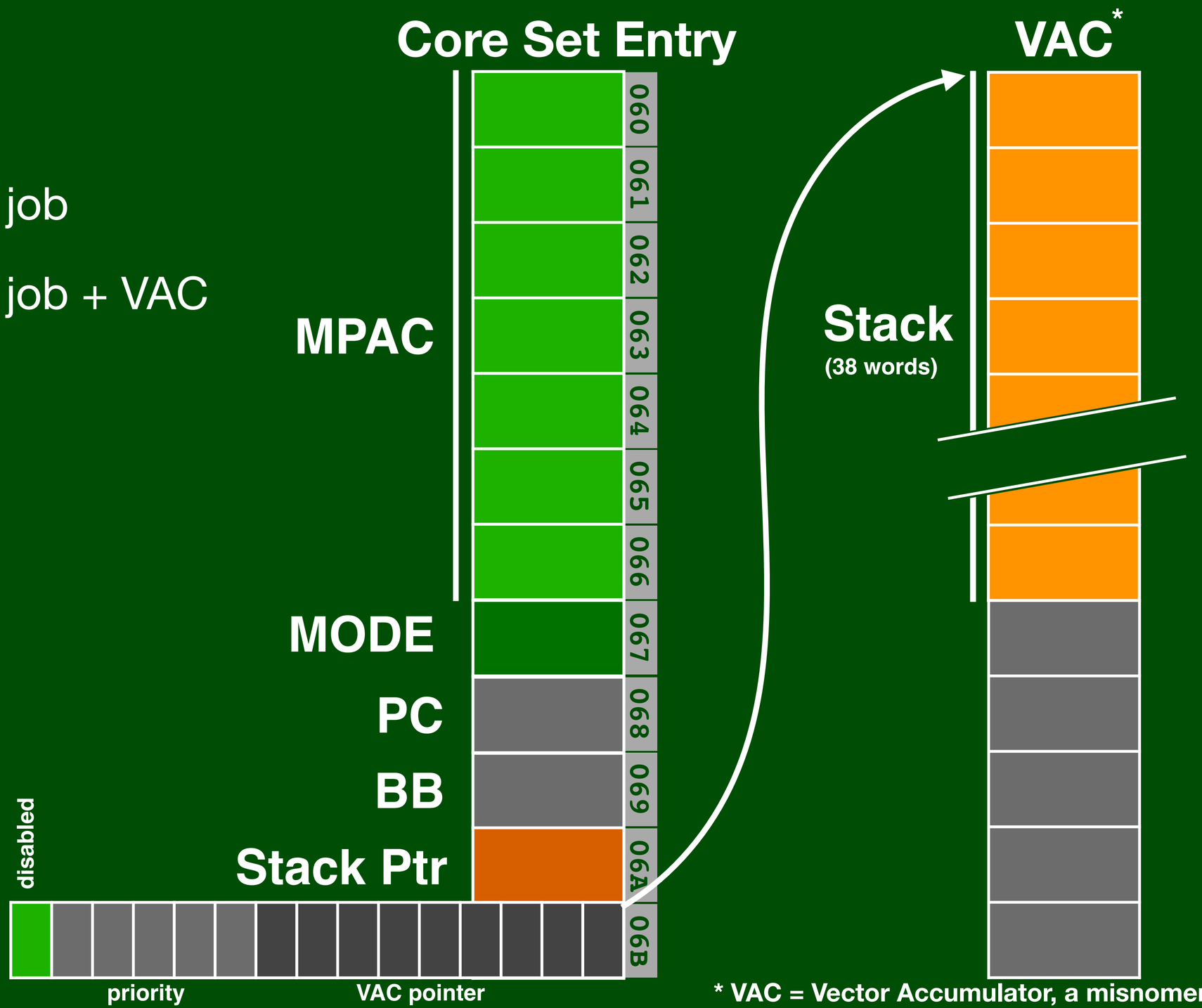


* VAC = Vector Accumulator, a misnomer

VAC

NOVAC Create new job

FINDVAC Create new job + VAC

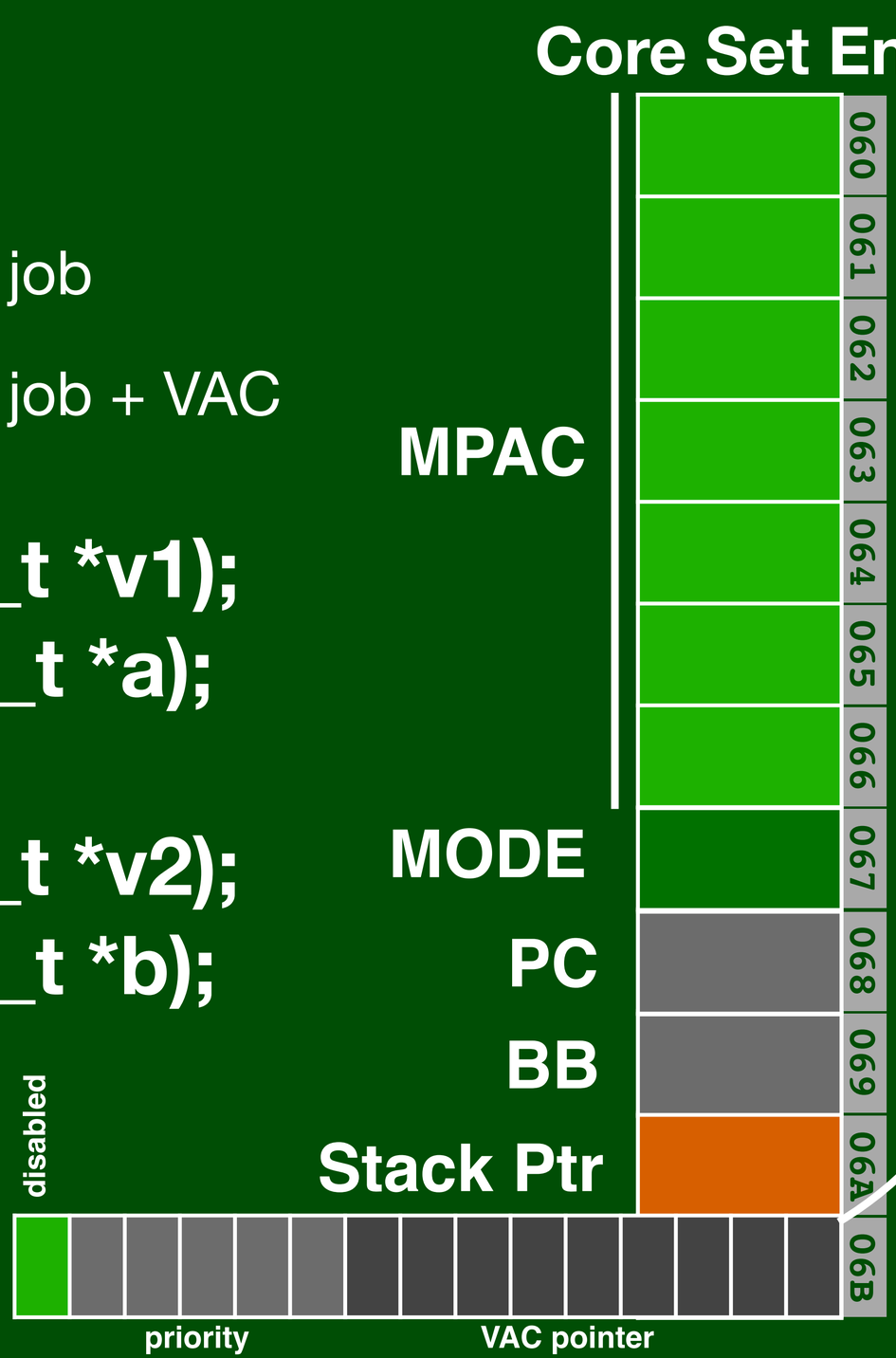


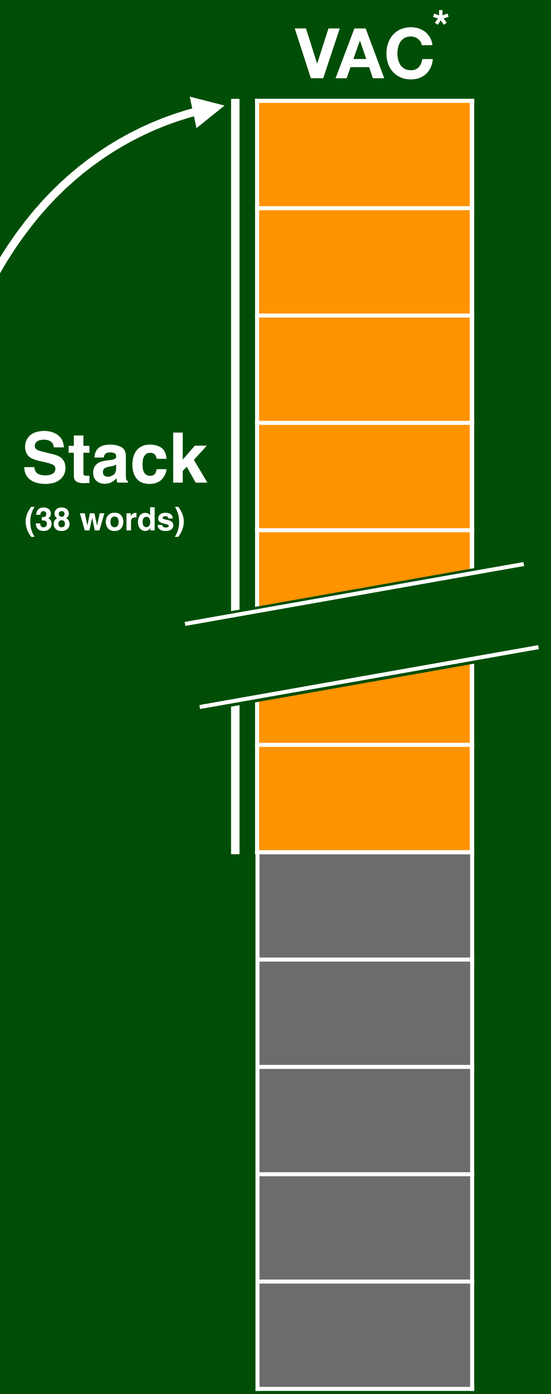
VAC

NOVAC Create new job

FINDVAC Create new job + VAC

```
load_vector(vector_t *v1);
mul_double(double_t *a);
push();
load_vector(vector_t *v2);
mul_double(double_t *b);
add_double(pop());
```





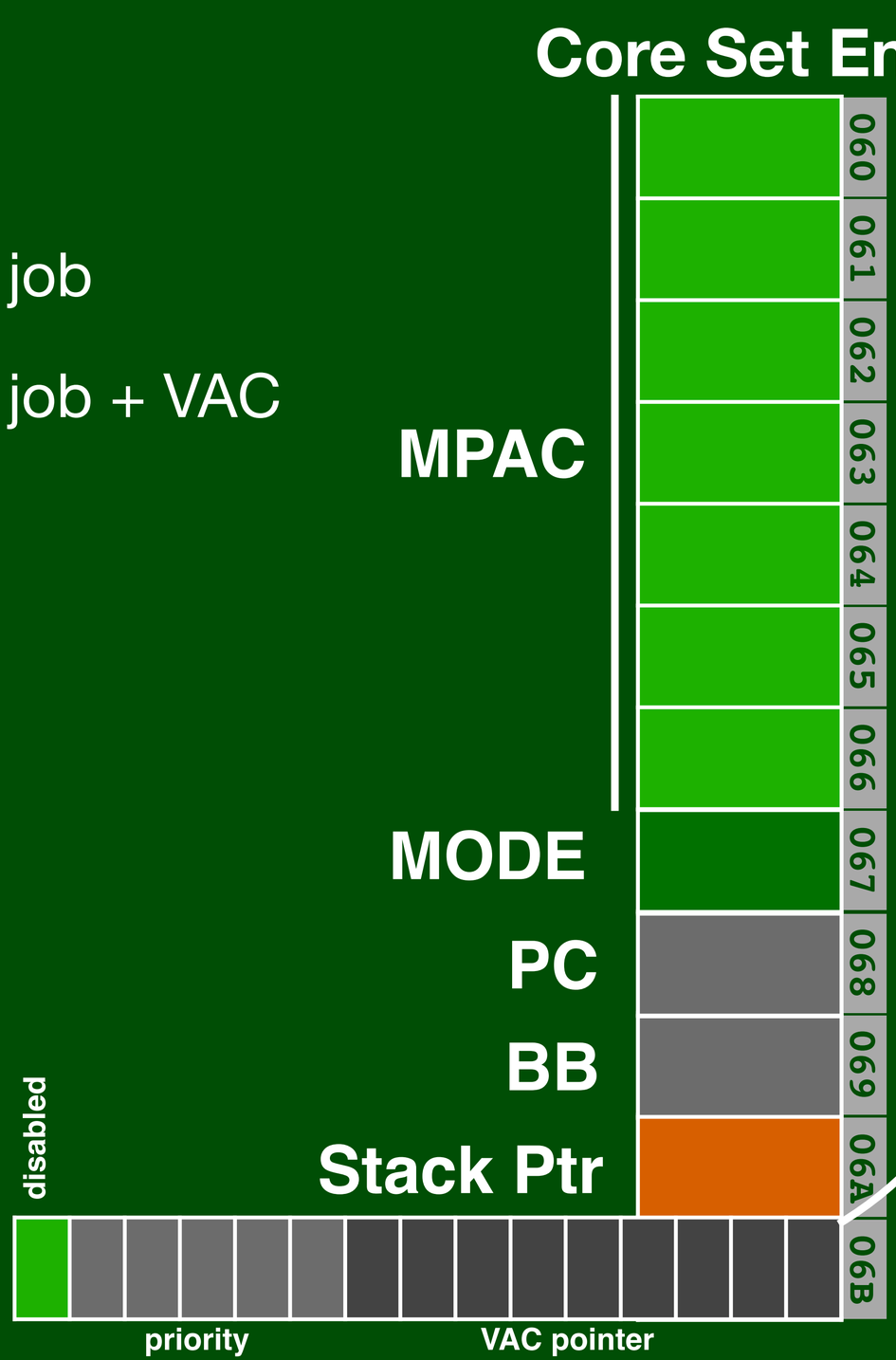
* VAC = Vector Accumulator, a misnomer

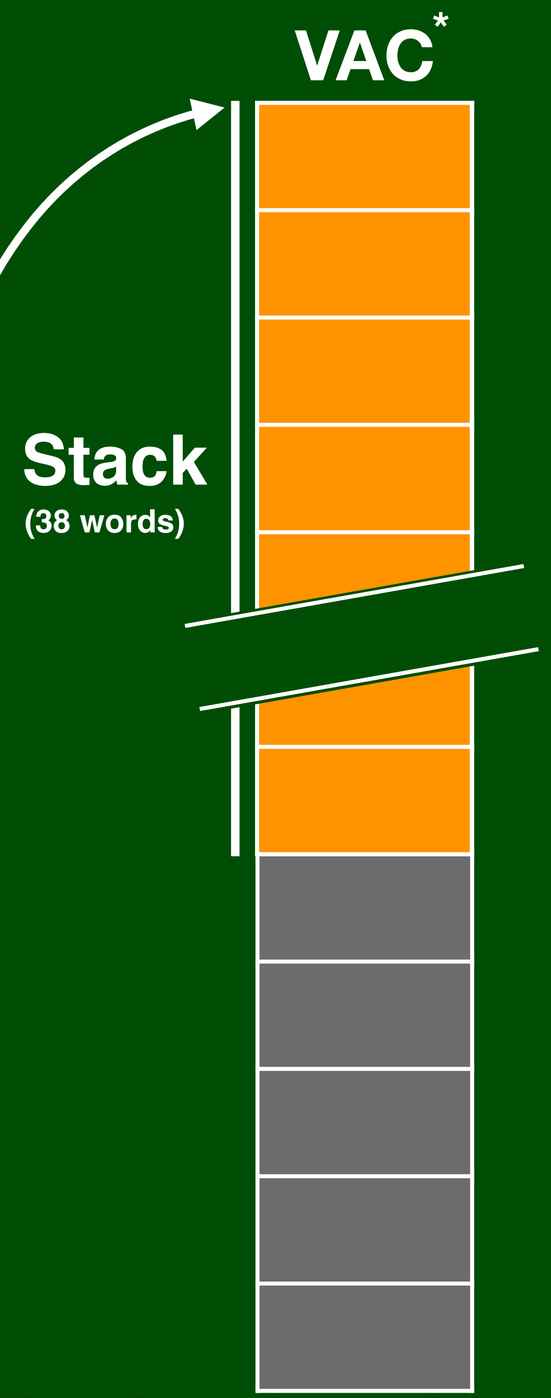
VAC

NOVAC Create new job

FINDVAC Create new job + VAC

```
VLOAD V1
DMP A
PUSH
VLOAD V2
DMP B
DAD
```

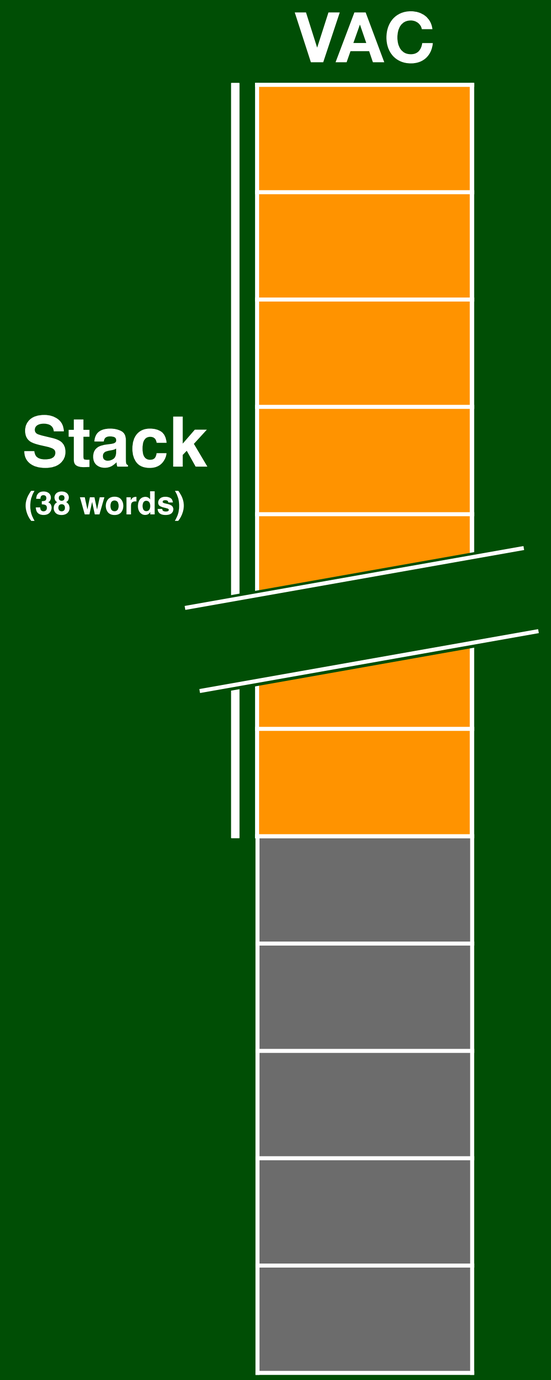




* VAC = Vector Accumulator, a misnomer

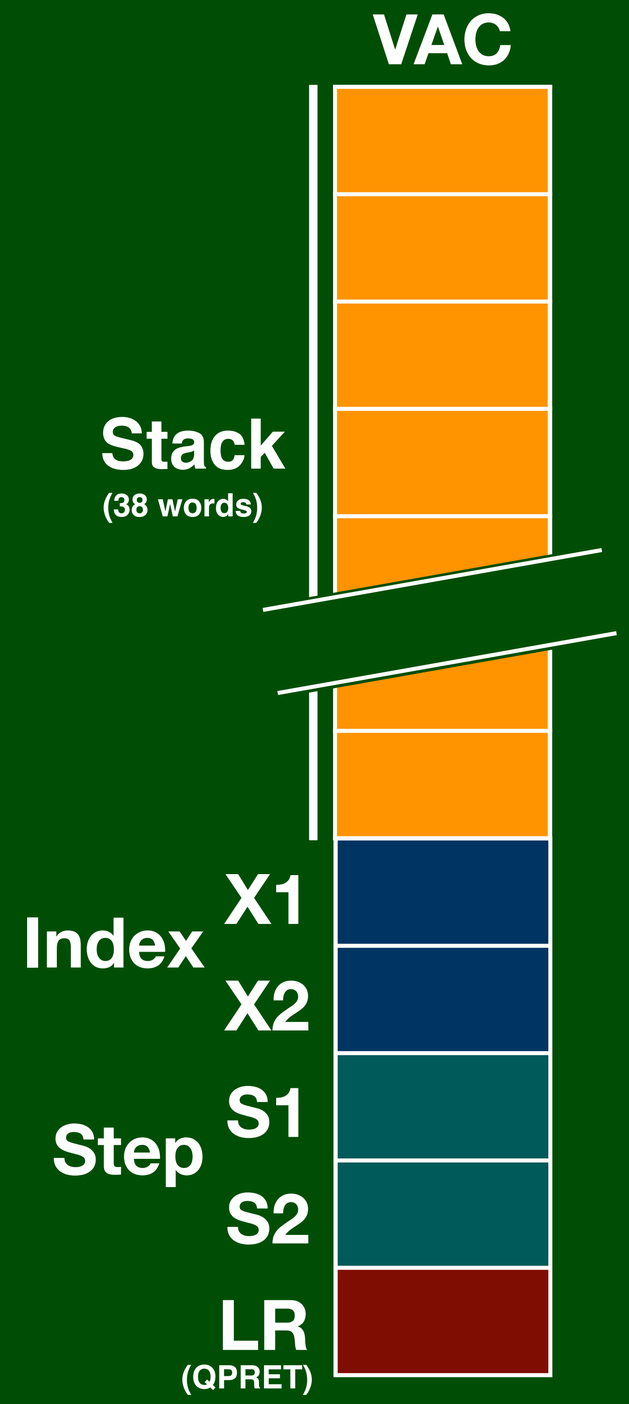
Interpretive Code

```
VLOAD V1
DMP A
PUSH
VLOAD V2
DMP B
DAD
```

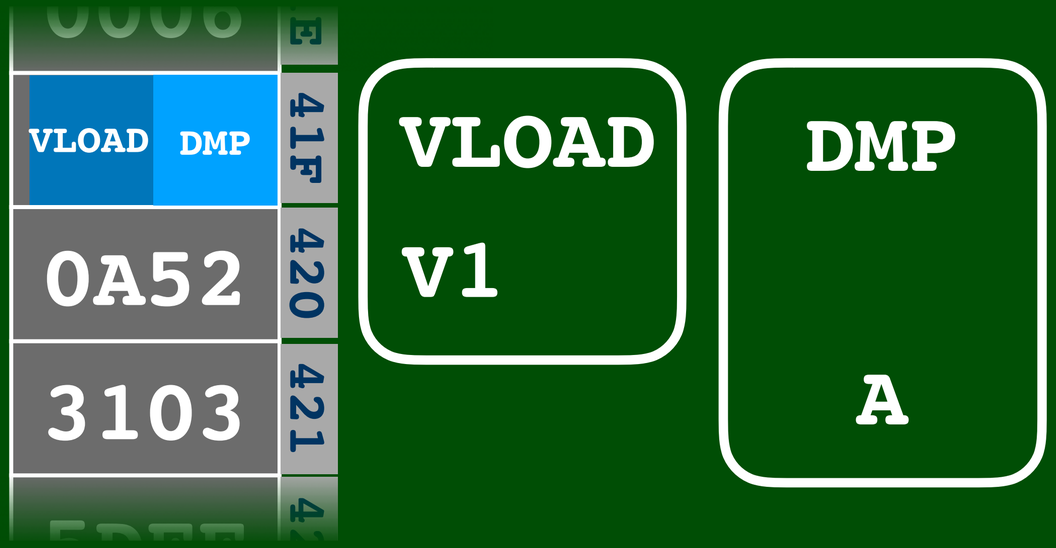


Interpretive Code

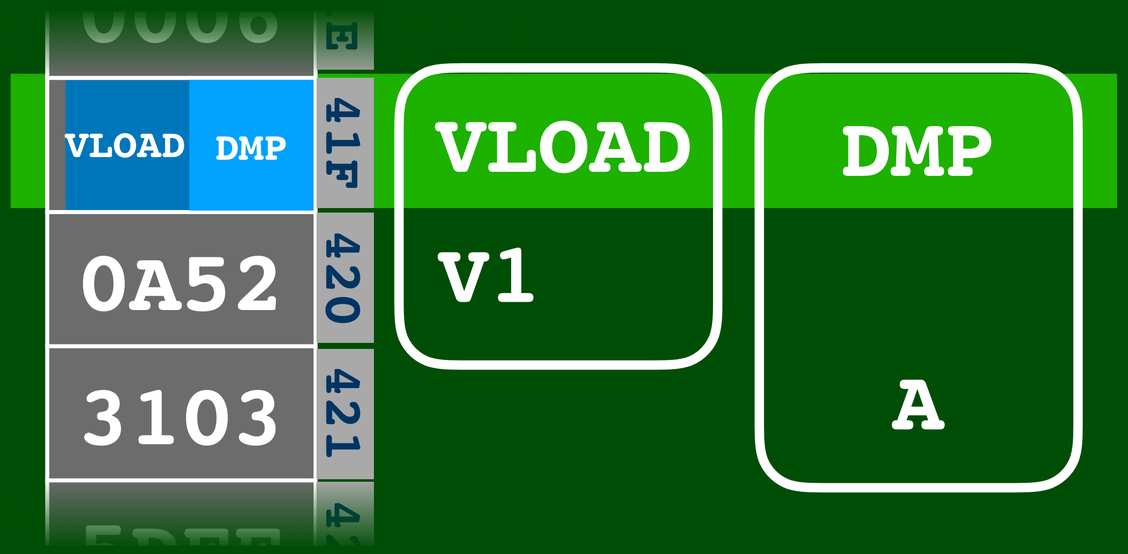
```
VLOAD  V1
DMP  A
PUSH
VLOAD  V2
DMP  B
DAD
```



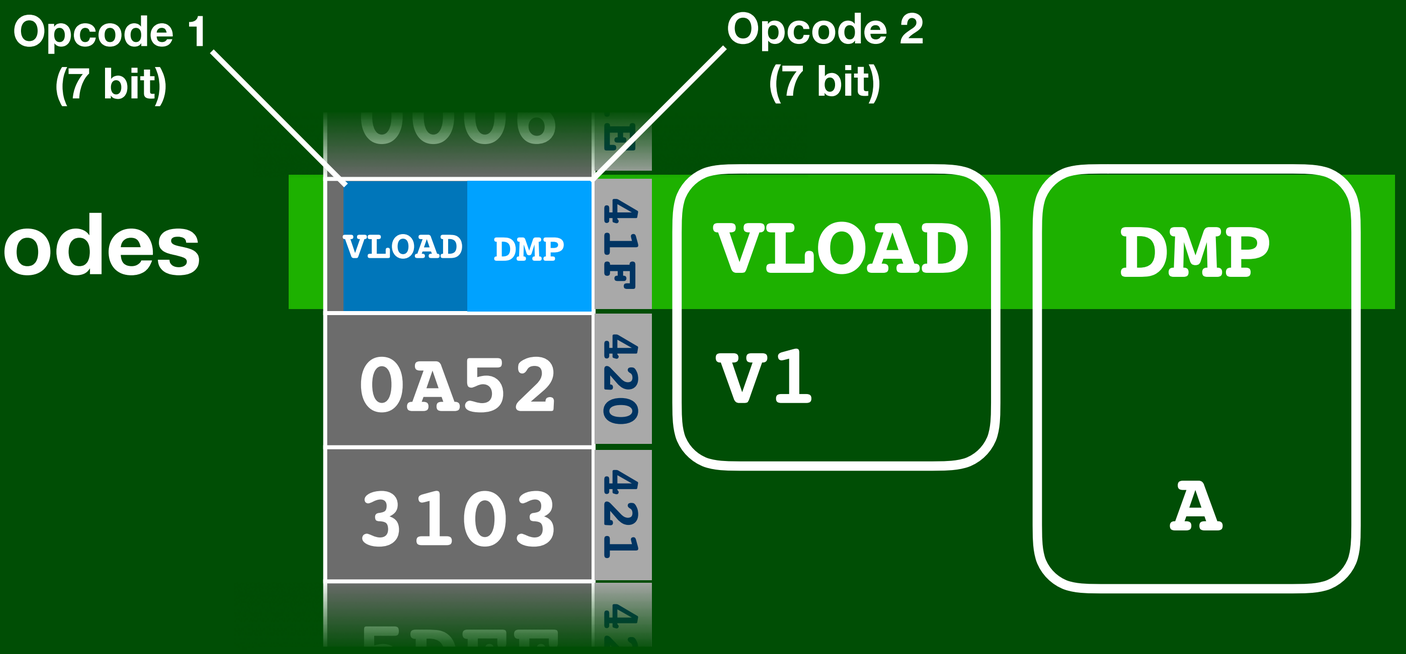
Interpreter Encoding



Interpreter Encoding

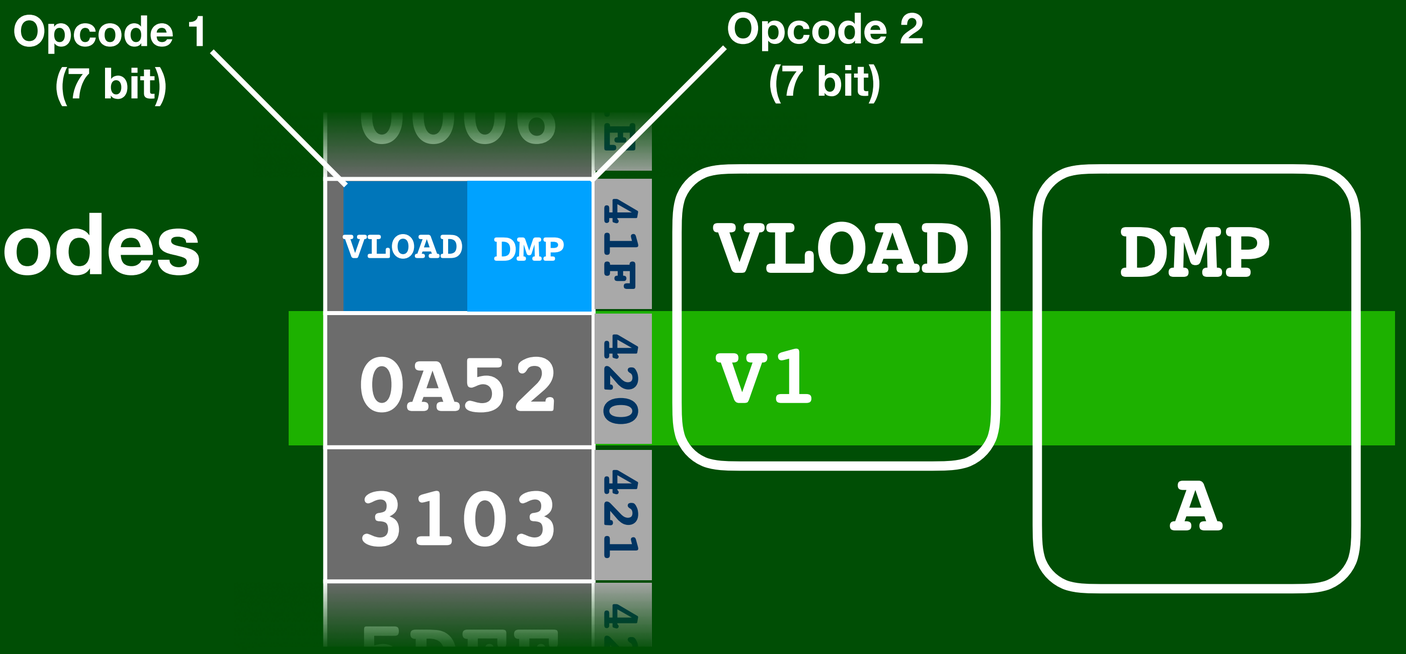


Interpreter Encoding



SHR7 0000 013

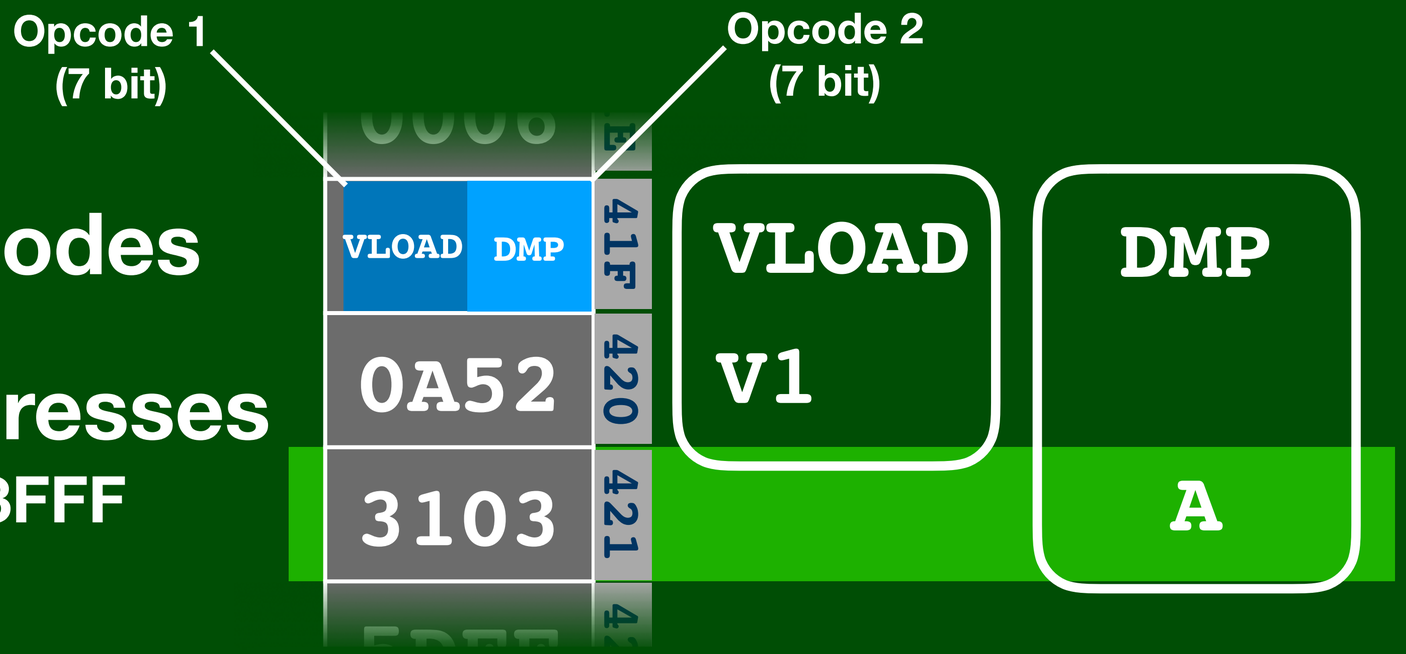
Interpreter Encoding



128 opcodes

SHR7 0000 013

Interpreter Encoding



128 opcodes

14 bit Addresses

$0000-$3FFF

SHR7 0000 013

Interpreter Memory

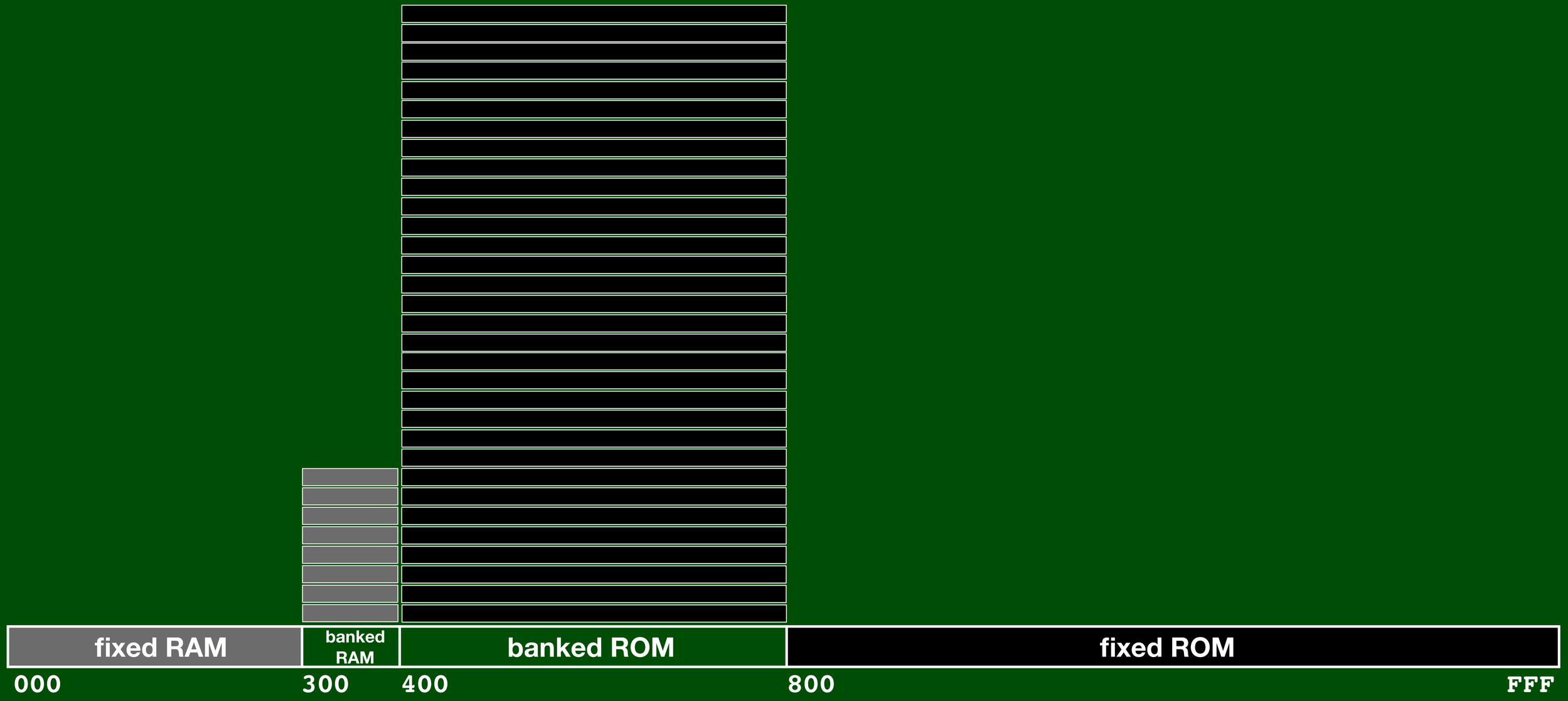
14 bit Addresses

$0000-$3FFF

Interpreter Memory

14 bit Addresses

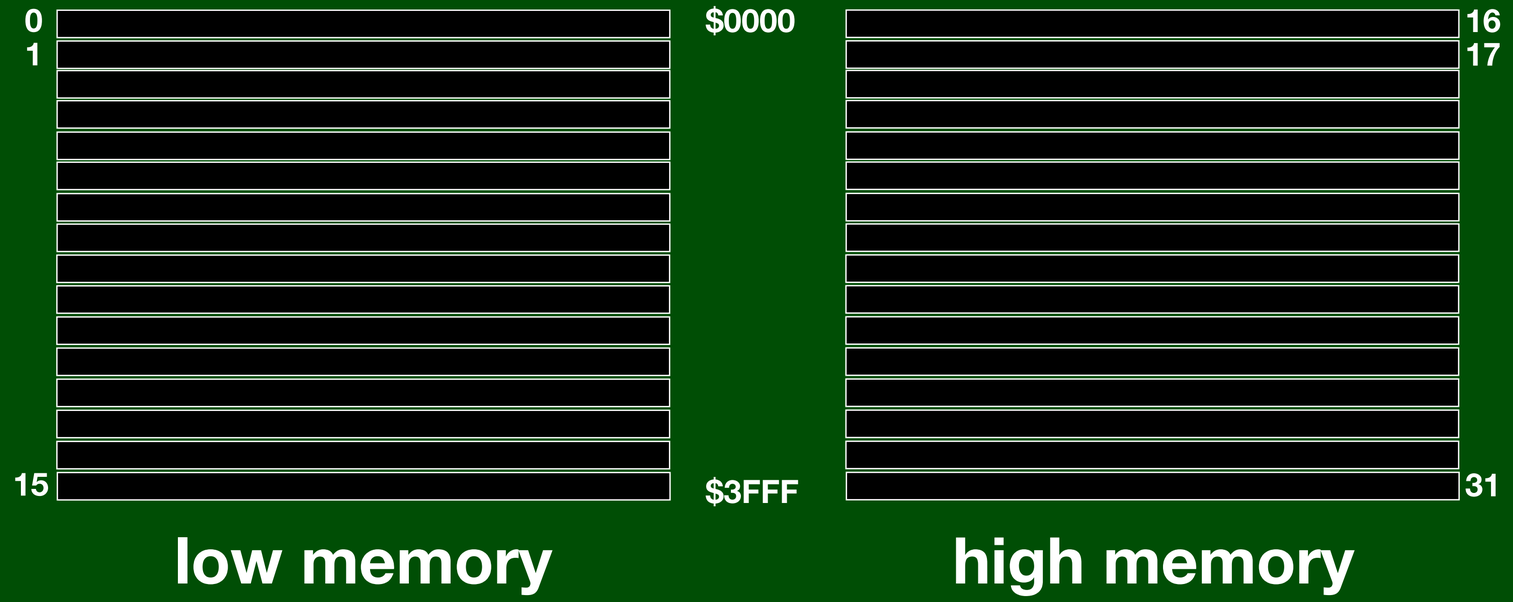
$0000-$3FFF



Interpreter Memory

14 bit Addresses

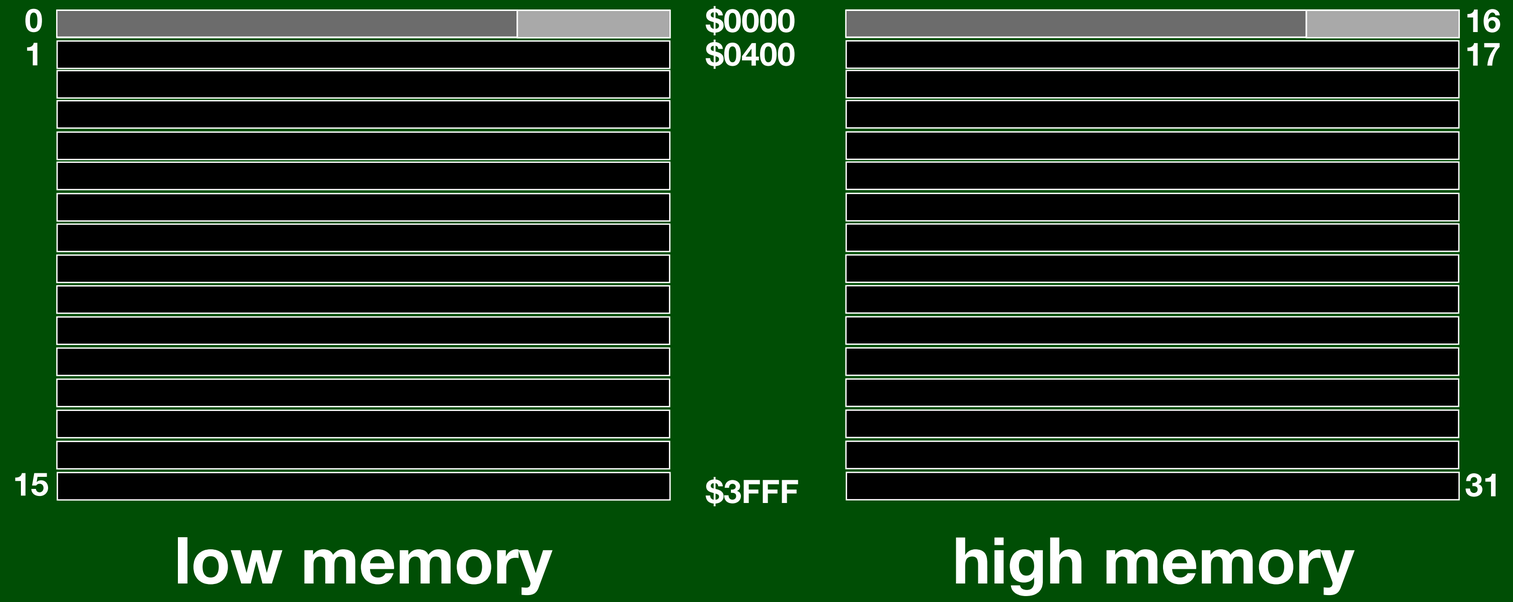
$0000-$3FFF



Interpreter Memory

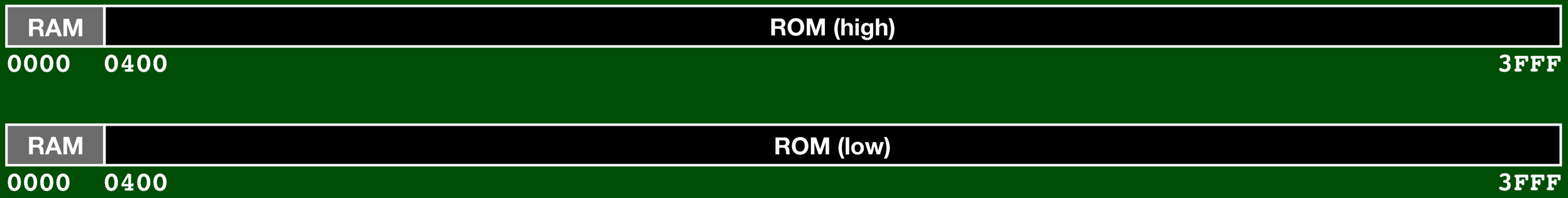
14 bit Addresses

$0000-$3FFF



Interpreter Memory

Interpreter Memory



Interpreter Instructions

Interpreter Instructions

Load/Store

SLOAD
DLOAD
TLOAD
VLOAD

STORE
STODL
STOVL
STCALL

Arithmetic

DAD
DSU
BDSU
DMP
DMPR
DDV
BDDV
SIGN
TAD

SQRT
SIN
COS
ARCSIN
ARCCOS
DSQ
ROUND
DCOMP
ABS

Vector

VAD
VSU
BVSU
DOT
VXSC
V/SC
VXV
VPROJ
VXM
MXV

Shift

SR
SRR
SL
SLR
VSR
VSL

Index

AXT
AXC
LXA
LXC
SXA
XCHX
SSP
INCR
XAD
XSU
TIX

Control Flow

GOTO
CALL
RVQ
STQ
CGOTO
CCALL
BPL
BZE
BMN
BHIZ
EXIT
RTB
BOV
BOVB

Stack

PUSH
PDDL
PDVL

Interpreter Instructions

Load/Store

- SLOAD
- DLOAD
- TLOAD
- VLOAD

- STORE
- STODL
- STOVL
- STCALL

Arithmetic

- DAD
- DSU
- BDSU
- DMP
- DMPR
- DDV
- BDDV
- SIGN
- TAD

- SQRT
- SIN
- COS
- ARCSIN
- ARCCOS
- DSQ
- ROUND
- DCOMP
- ABS

Vector

- VAD
- VSU
- BVSU
- DOT
- VXSC
- V/SC
- VXV
- VPROJ
- VXM
- MXV

Shift

- SR
- SRR
- SL
- SLR
- VSR
- VSL

Index

- AXT
- AXC
- LXA
- LXC
- SXA
- XCHX
- SSP
- INCR
- XAD
- XSU
- TIX

Control Flow

- GOTO
- CALL
- RVQ
- STQ
- CGOTO
- CCALL
- BPL
- BZE
- BMN
- BHIZ
- EXIT
- RTB
- BOV
- BOVB

Stack

- PUSH
- PDDL
- PDVL

Interpreter Instructions

Load/Store

- SLOAD
- DLOAD
- TLOAD
- VLOAD

- STORE
- STODL
- STOVL
- STCALL

Arithmetic

- DAD
- DSU
- BDSU
- DMP
- DMPR
- DDV
- BDDV
- SIGN
- TAD

- SQRT
- SIN
- COS
- ARCSIN
- ARCCOS
- DSQ
- ROUND
- DCOMP
- ABS

Vector

- VAD
- VSU
- BVSU
- DOT
- VXSC
- V/SC
- VXV
- VPROJ
- VXM
- MXV

Shift

- SR
- SRR
- SL
- SLR
- VSR
- VSL

Index

- AXT
- AXC
- LXA
- LXC
- SXA
- XCHX
- SSP
- INCR
- XAD
- XSU
- TIX

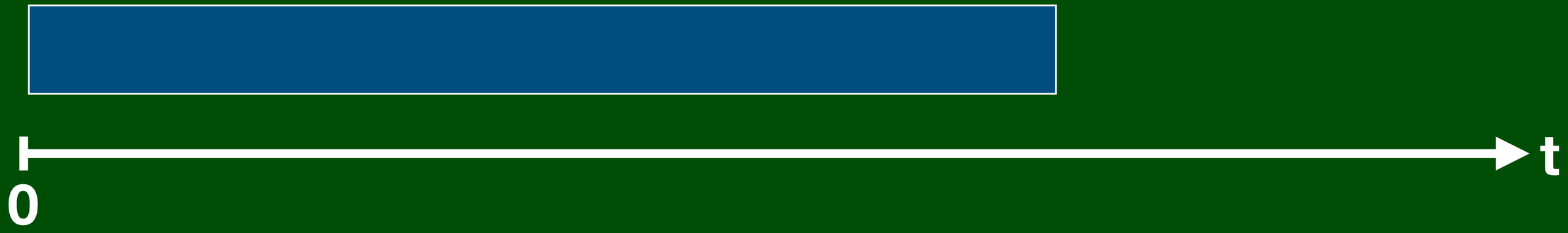
Control Flow

- GOTO
- CALL
- RVQ
- STQ
- CGOTO
- CCALL
- BPL
- BZE
- BMN
- BHIZ
- EXIT ⟷ CALL INTPRET
- RTB
- BOV
- BOVB

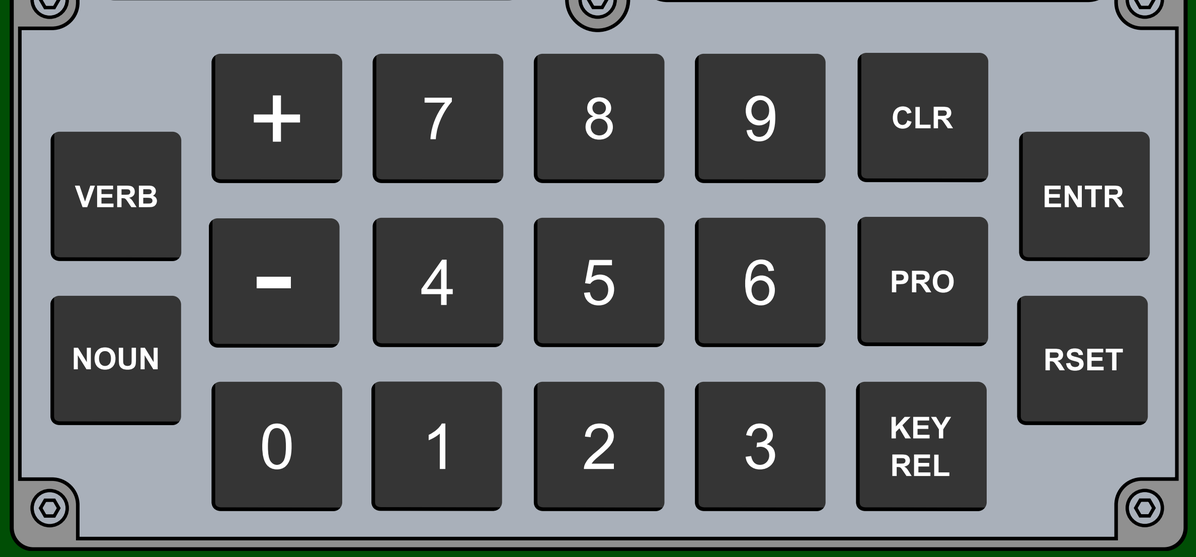
Stack

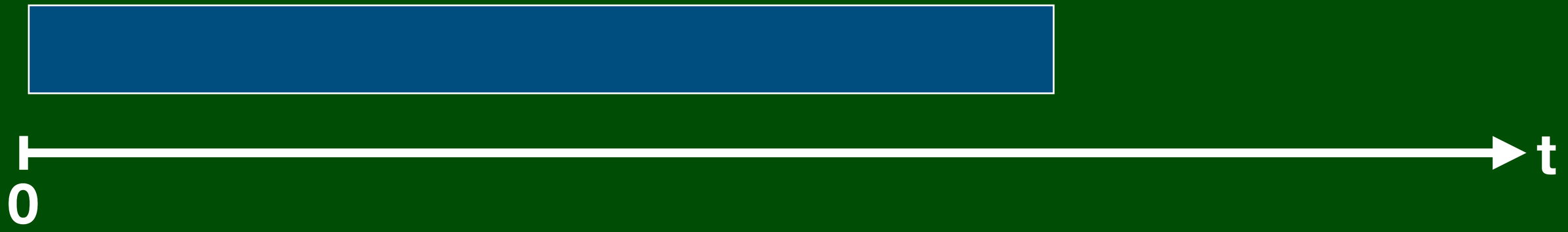
- PUSH
- PDDL
- PDVL

Interrupts

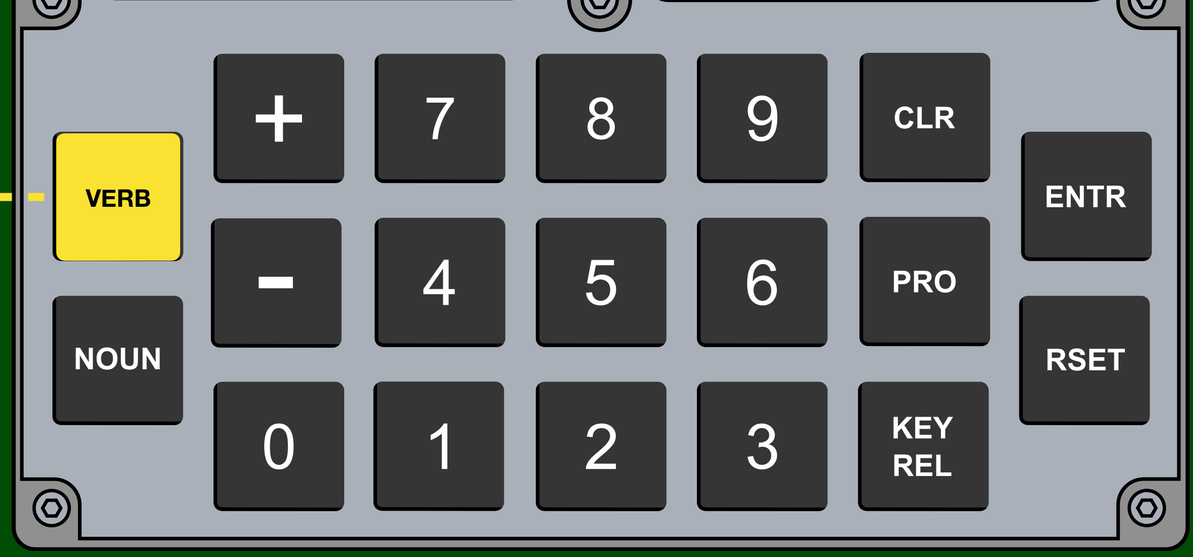


Interrupts



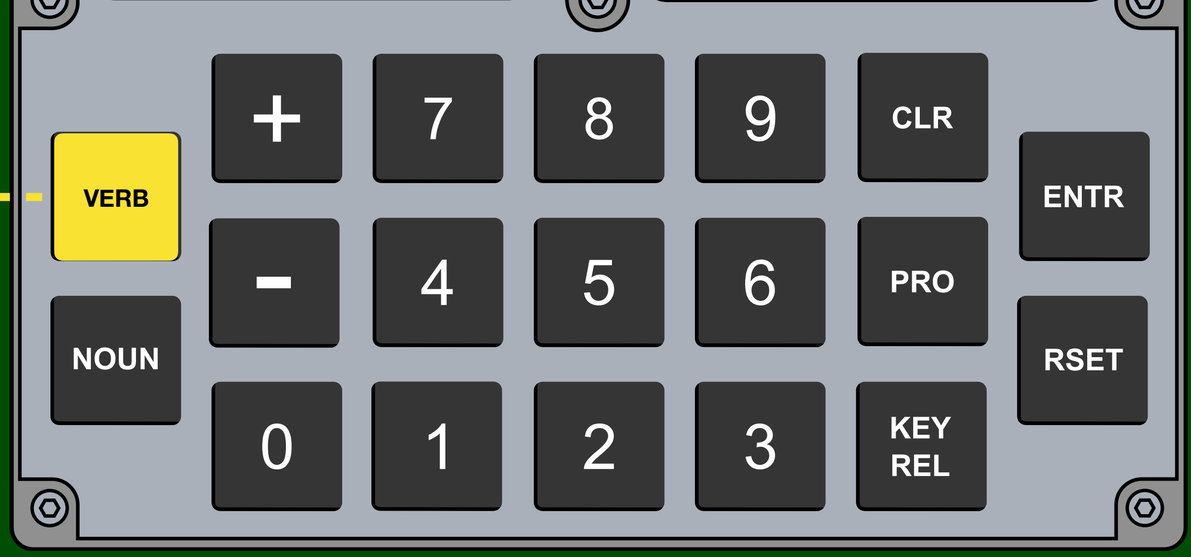


Interrupts



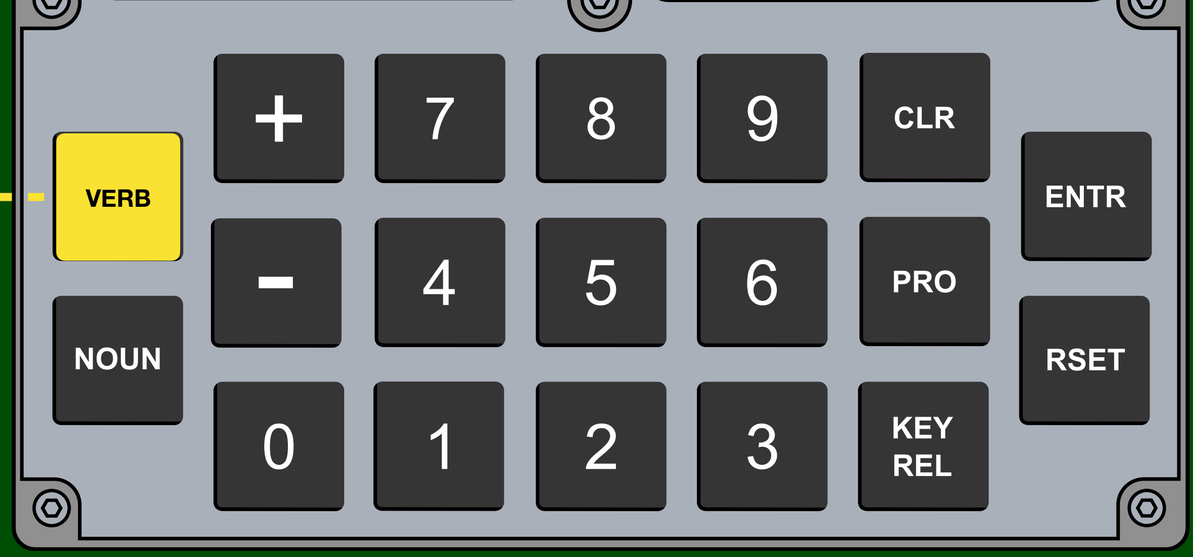
Interrupt

Interrupts



Interrupt

Interrupts



Interrupt

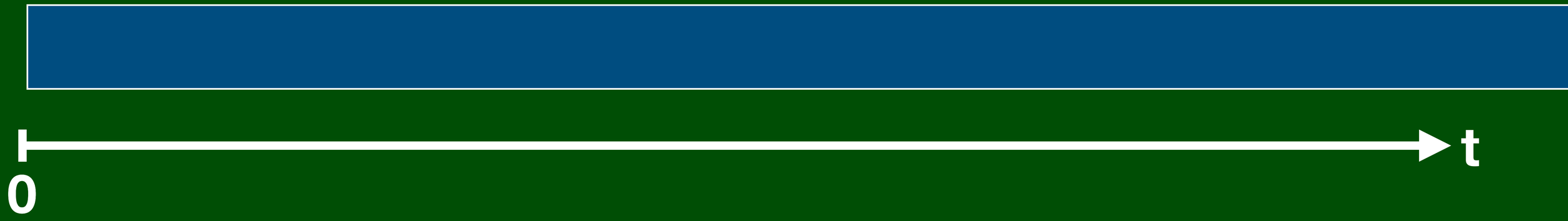
NOVAC
create job

CHARIN

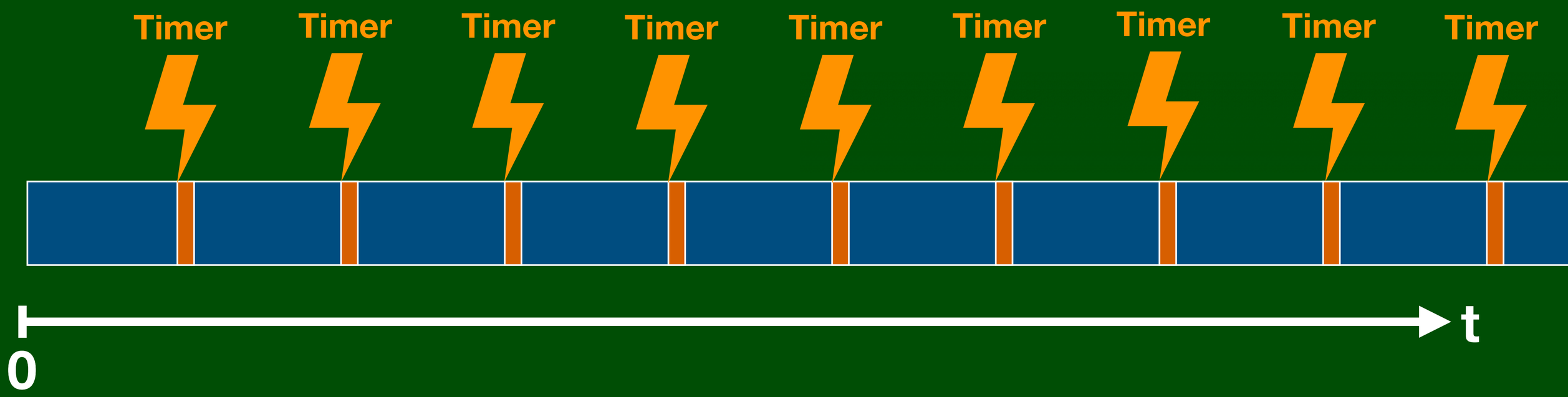
0

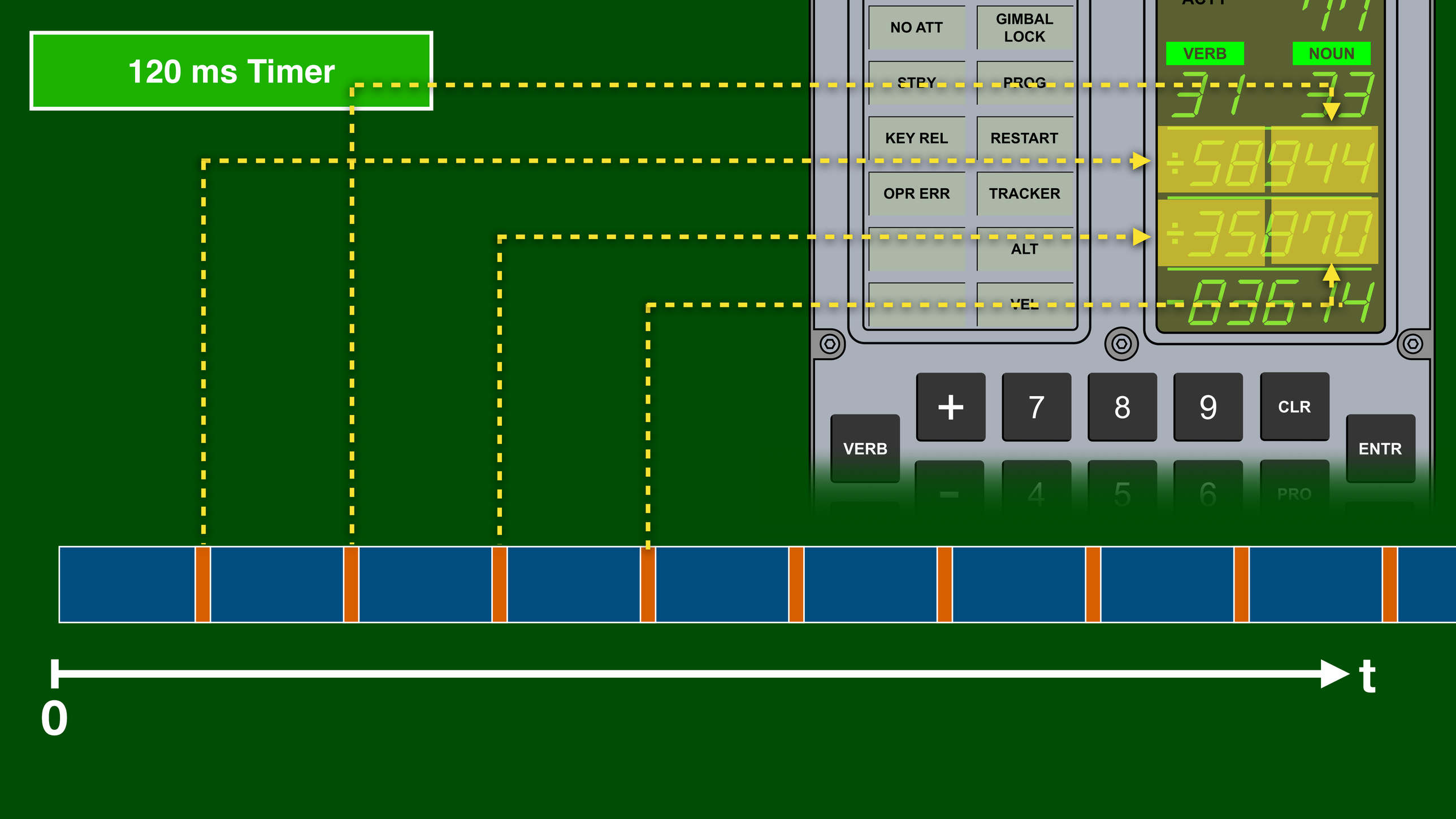
t

120 ms Timer



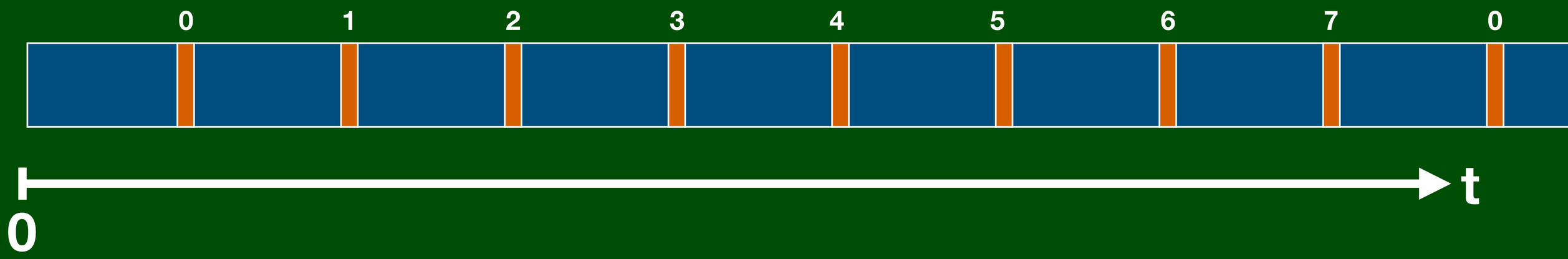
120 ms Timer




120 ms Timer
NO ATT
GIMBAL LOCK
STBY
PROG
KEY REL
RESTART
OPR ERR
TRACKER
ALT
VEL
VERB
NOUN
VERB
+
7
8
9
CLR
ENTR
0
t

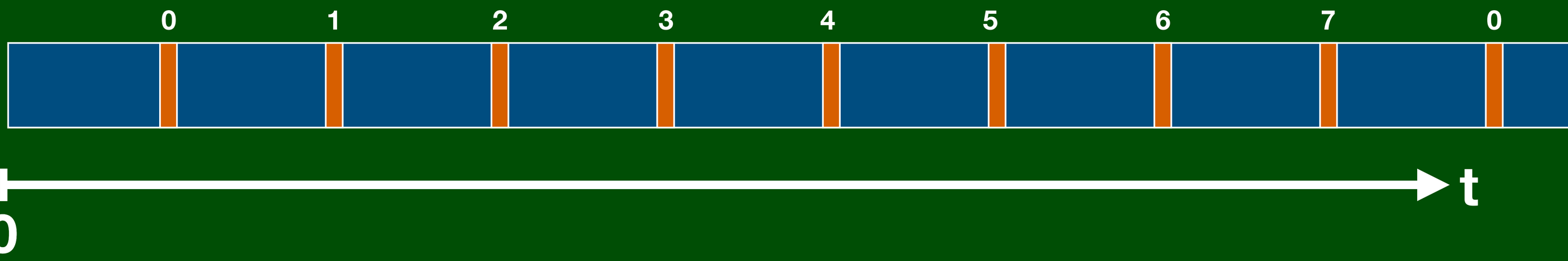
Drivers

Drivers



Drivers

CM	OPTDRIVE OPTTEST		OPTMON			OPTDRIVE OPTTEST	OPTMON	
LM	DSKY	DSKY	IMUMON DSKY	DSKY	DSKY	DSKY	IMUMON DSKY	DSKY
	RCSMONIT	DAPT4S RRAUTCHK		DAPT4S	RCSMONIT	DAPT4S RRAUTCHK		DAPT4S



Waitlist Tasks

t

Waitlist Tasks

Job A

t

Waitlist Tasks

Job A

t

Waitlist Tasks

Task 1 @ 0.6

Task 2 @ 0.8

Task 3 @ 1.4

Job A

1

2

3

t

t_0

t_0 + 0.6s

t_0 + 0.8s

t_0 + 1.4s

Waitlist Tasks

Task 1 @ 0.6

Task 2 @ 0.8

Task 3 @ 1.4

Job A

1

2

3

t

t_0

t_0 + 0.6s

t_0 + 0.8s

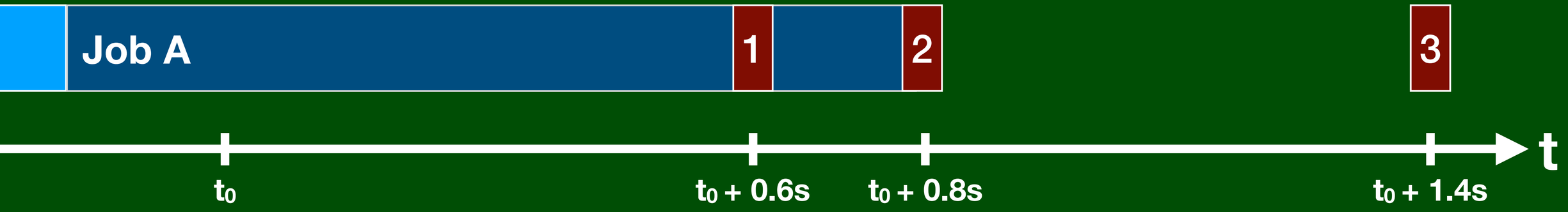
t_0 + 1.4s

Waitlist Tasks

Task 1 @ 0.6

Task 2 @ 0.8

Task 3 @ 1.4

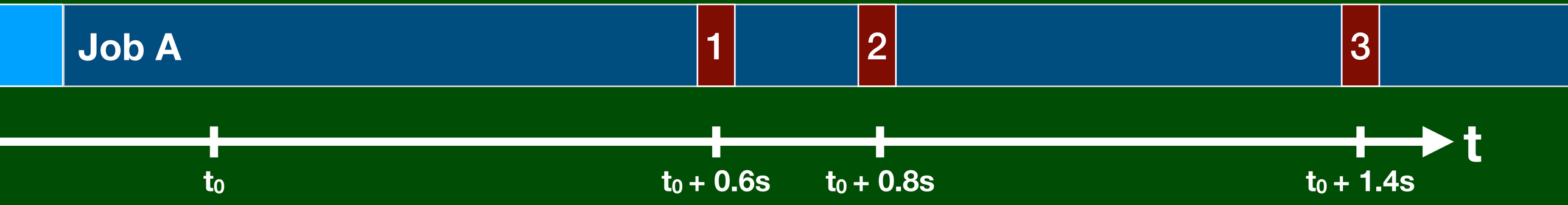


Waitlist Tasks

Task 1 @ 0.6

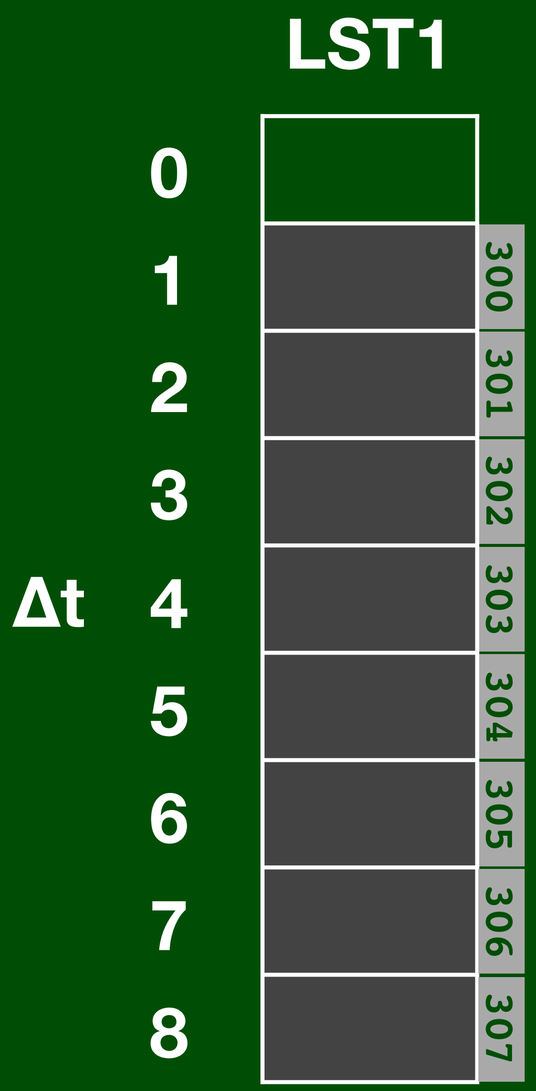
Task 2 @ 0.8

Task 3 @ 1.4

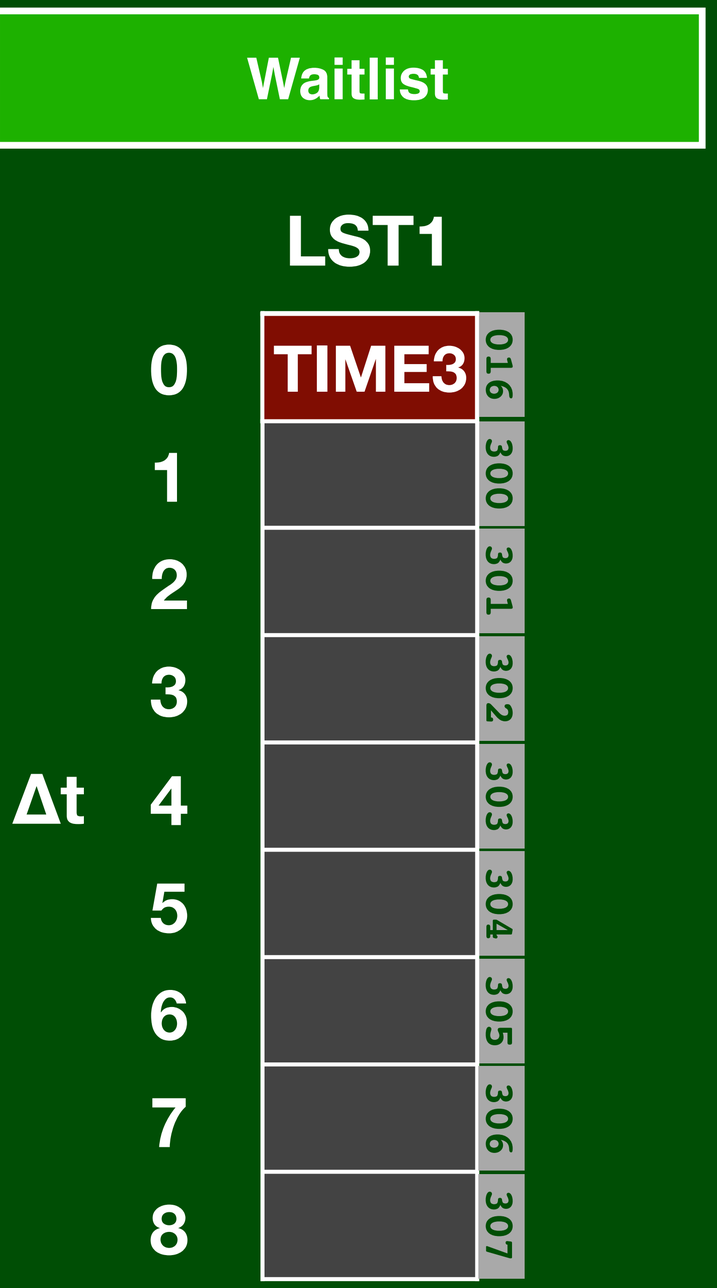


Waitlist

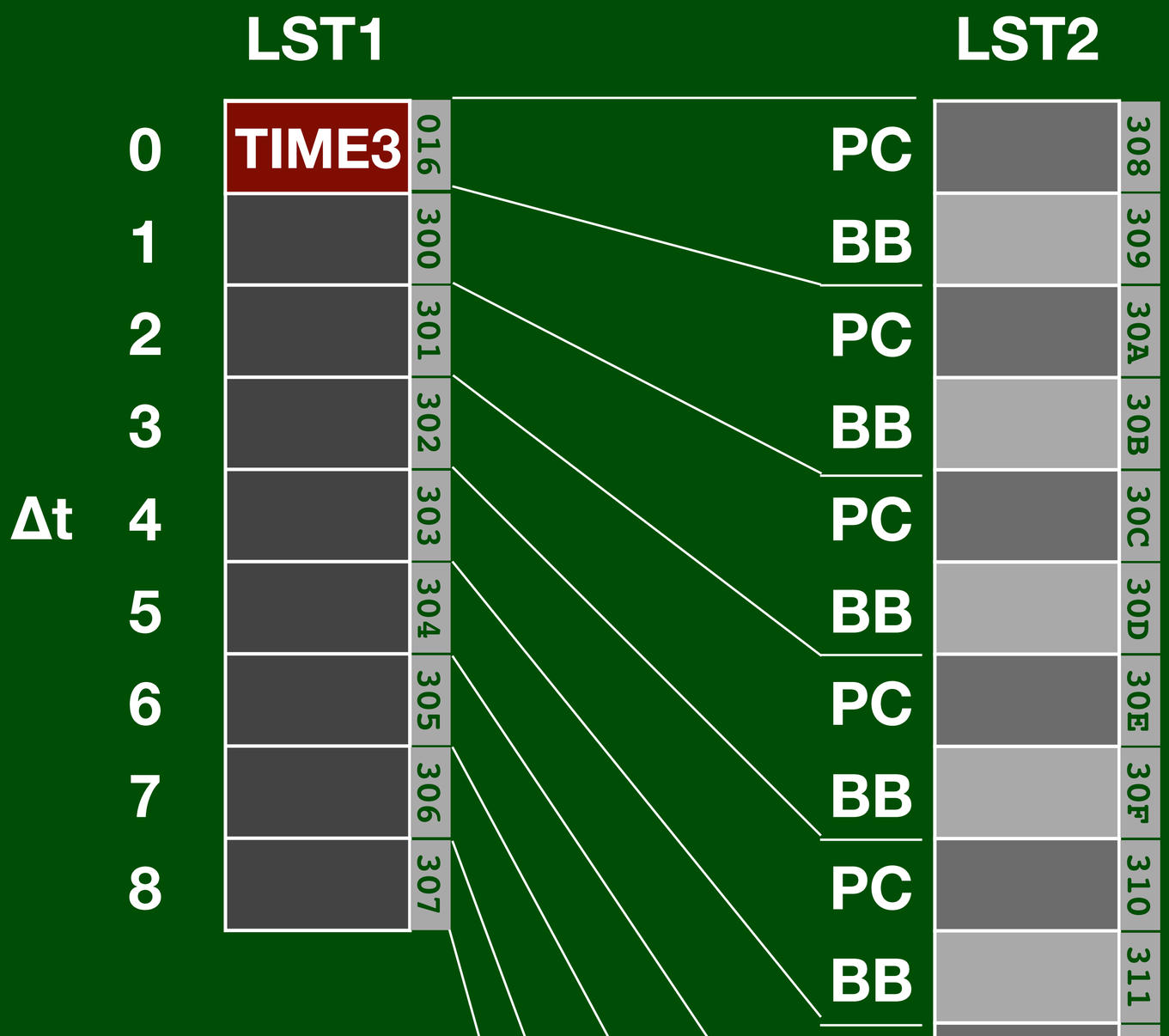
Waitlist



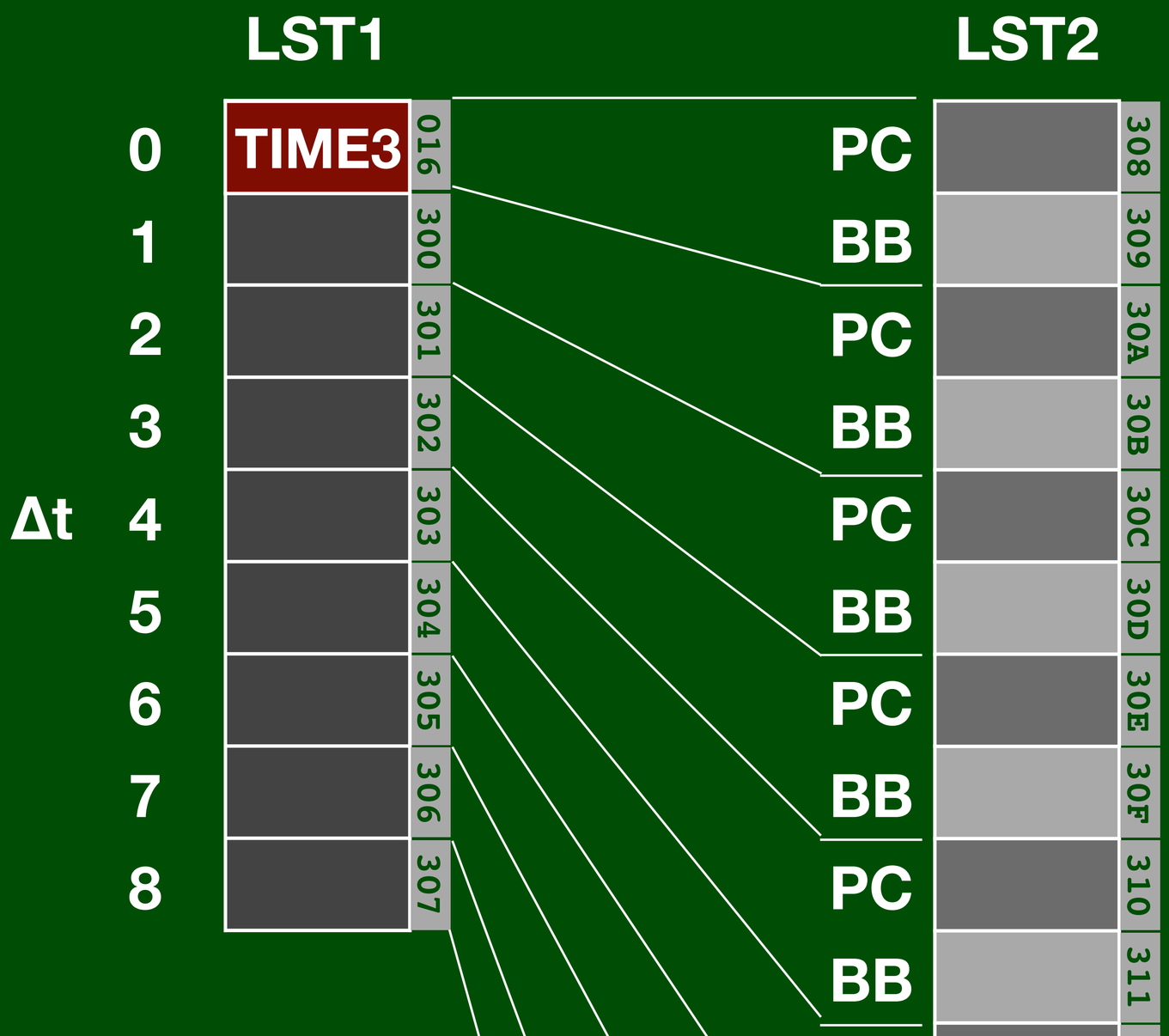
Waitlist



Waitlist


LST1
LST2
Δt
0
1
2
3
4
5
6
7
8
TIME3
016
300
301
302
303
304
305
306
307
PC
BB
PC
BB
PC
BB
PC
BB
PC
BB
308
309
30A
30B
30C
30D
30E
30F
310
311

Waitlist



WAITLIST Create new task

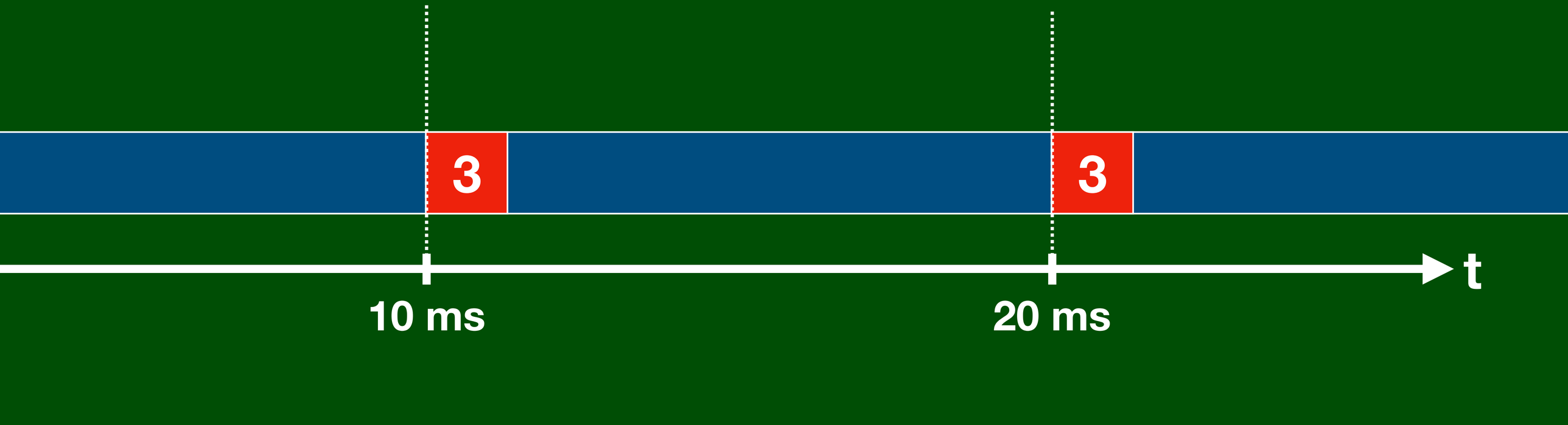
TASKOVER End current task

Real-Time

Real-Time

every 10 ms

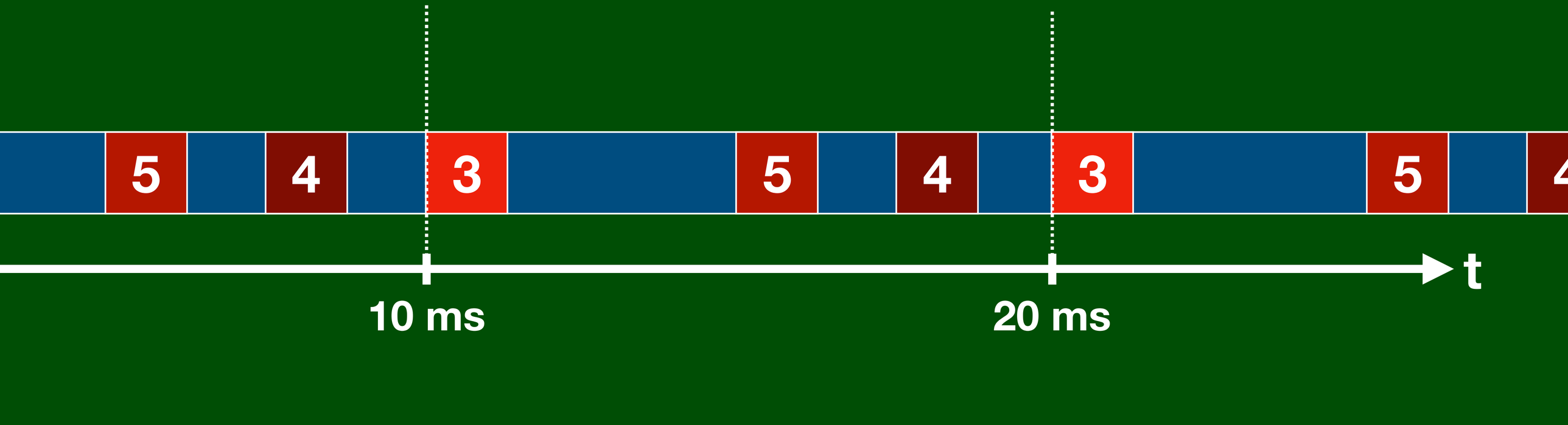
waitlist TIME3 0000 016 + 0 ms



Real-Time

every 10 ms

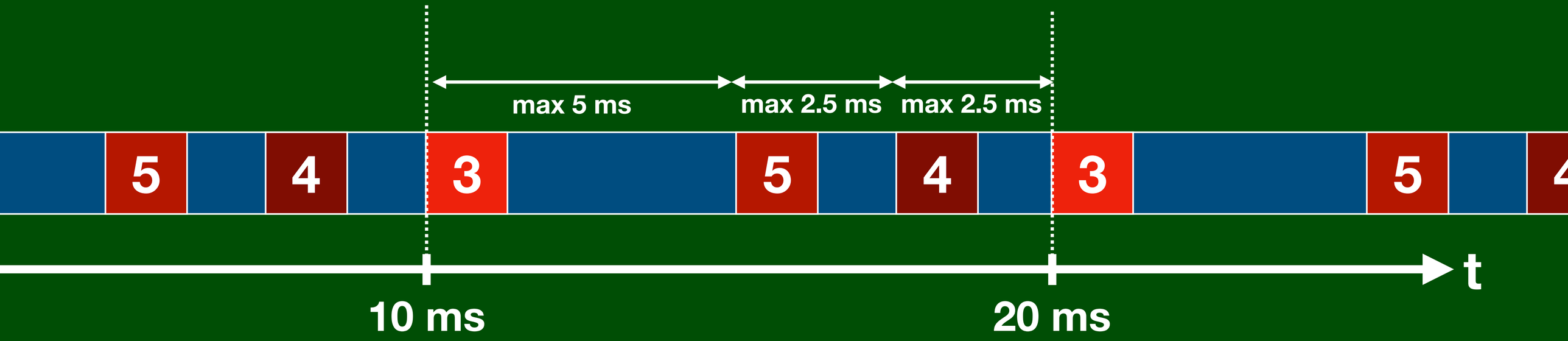
waitlist	TIME3	0000	016	+ 0 ms
autopilot	TIME5	0000	018	+ 5 ms
drivers	TIME4	0000	017	+ 7.5 ms



Real-Time

every 10 ms

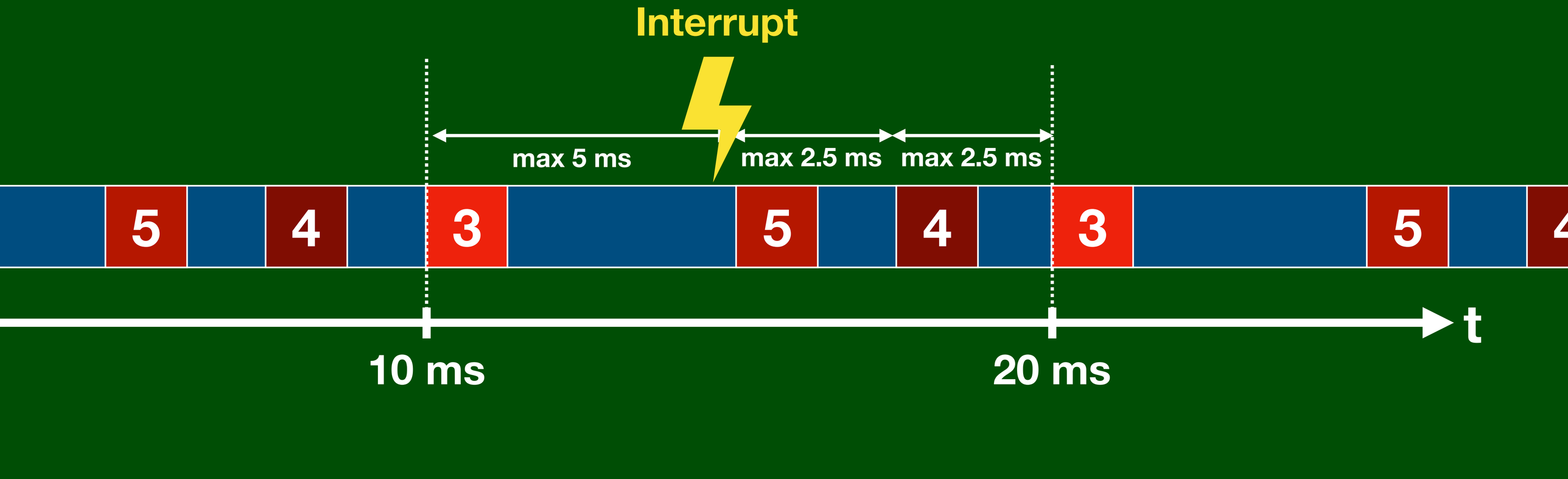
waitlist	TIME3	0000	016	+ 0 ms
autopilot	TIME5	0000	018	+ 5 ms
drivers	TIME4	0000	017	+ 7.5 ms



Real-Time

every 10 ms

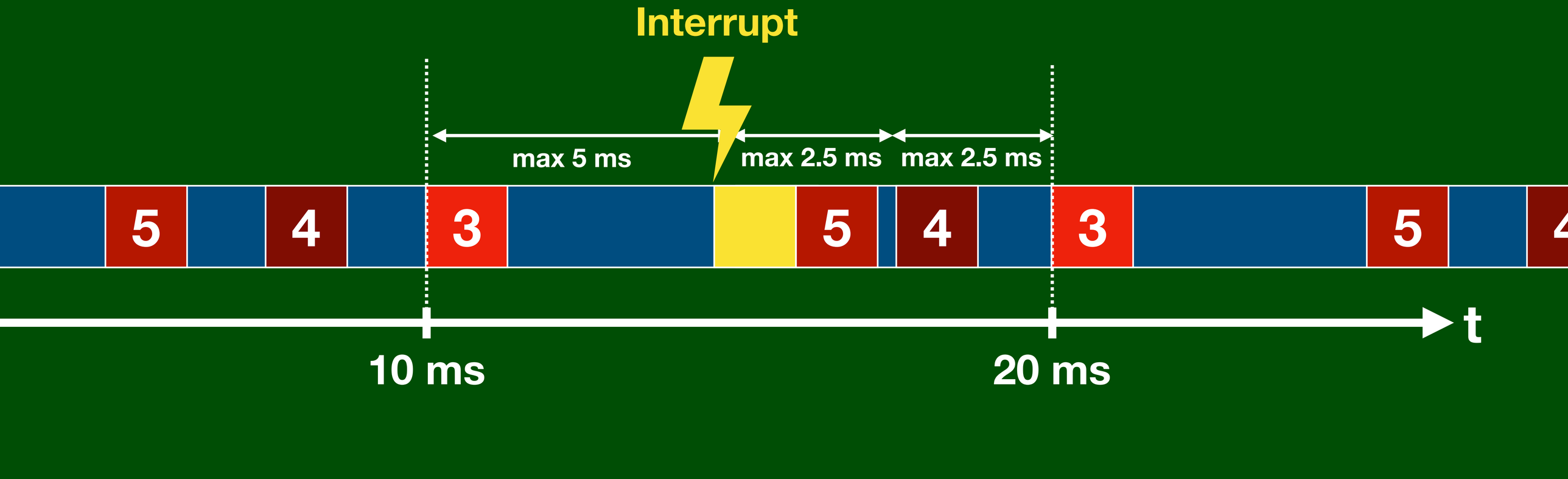
waitlist	TIME3	0000	016	+ 0 ms
autopilot	TIME5	0000	018	+ 5 ms
drivers	TIME4	0000	017	+ 7.5 ms



Real-Time

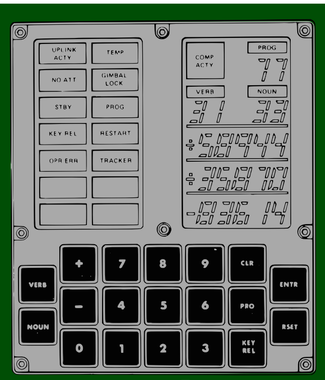
every 10 ms

waitlist	TIME3	0000	016	+ 0 ms
autopilot	TIME5	0000	018	+ 5 ms
drivers	TIME4	0000	017	+ 7.5 ms



PINBALL – the Shell

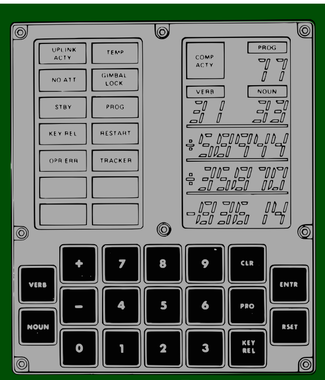
PINBALL – the Shell



keypress

Inter rupt

PINBALL – the Shell



keypress

Inter rupt

update state

CHARIN

display

VERBREG

NOUNREG

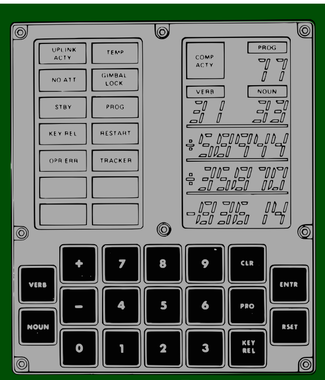
024

025

DSPTAB

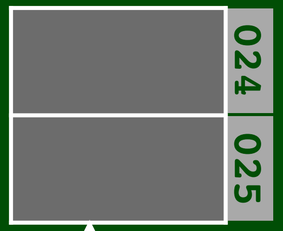
212 213 214 215 215 216 217 218 219 21A 21B

PINBALL – the Shell

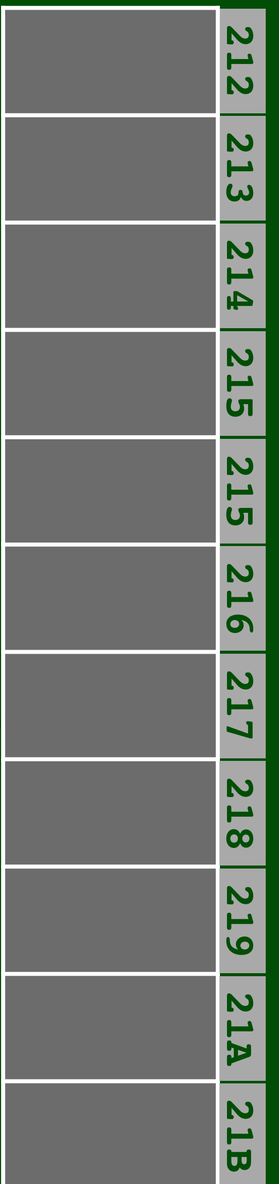


VERBREG

NOUNREG



DSPTAB



keypress

Inter rupt

update state

CHARIN

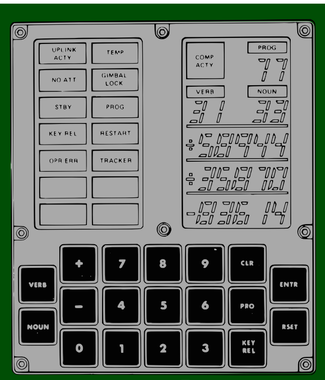
display

120 ms timer

Timer rupt

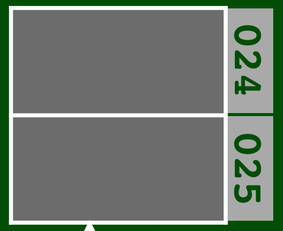
output

PINBALL – the Shell

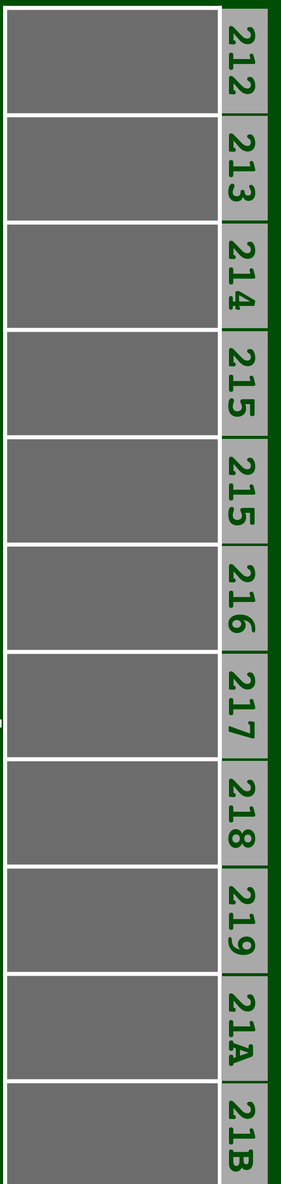


VERBREG

NOUNREG



DSPTAB



keypress

Inter rupt

update state

CHARIN

display

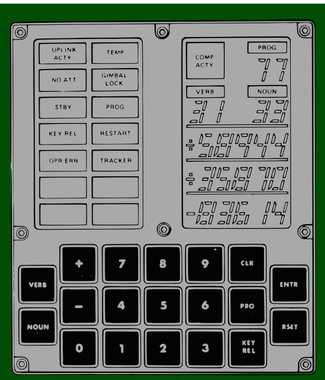
execute

120 ms timer

Timer rupt

output

PINBALL – the Shell



VERBREG

NOUNREG

024

025

DSPTAB

212 213 214 215 215 216 217 218 219 21A 21B

keypress

Inter rupt

uplink

Inter rupt

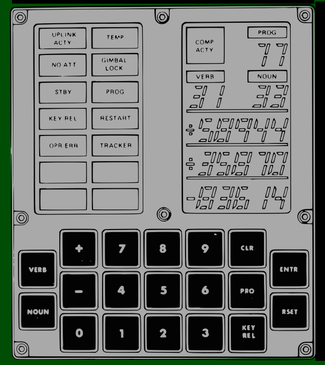
update state

CHARIN

display

execute

Mission Control



Radio

120 ms timer

Timer rupt

output

20 ms timer

Down rupt

downlink

Program Alarm

Program Alarm

0B77	41F
3282	420

```
call ALARM
.word 031202
```

Program Alarm

0B77	41F
3282	420

```
call ALARM
.word 031202
```

Program Abort

Program Abort

Interrupt Watchdog

JMP Watchdog

NEWJOB Watchdog

Parity


```
call ABORT
.word 01102
```

Program Abort

Interrupt Watchdog

JMP Watchdog

NEWJOB Watchdog

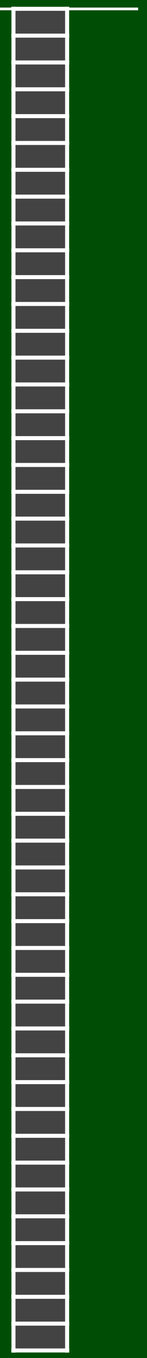
Parity

RESET


```
call ABORT
.word 01102
```

Program Abort

Job 4



Interrupt Watchdog

JMP Watchdog

NEWJOB Watchdog

Parity

RESET

```
0B77  41F  call ABORT
0242  420  .word 01102
```

Program Abort

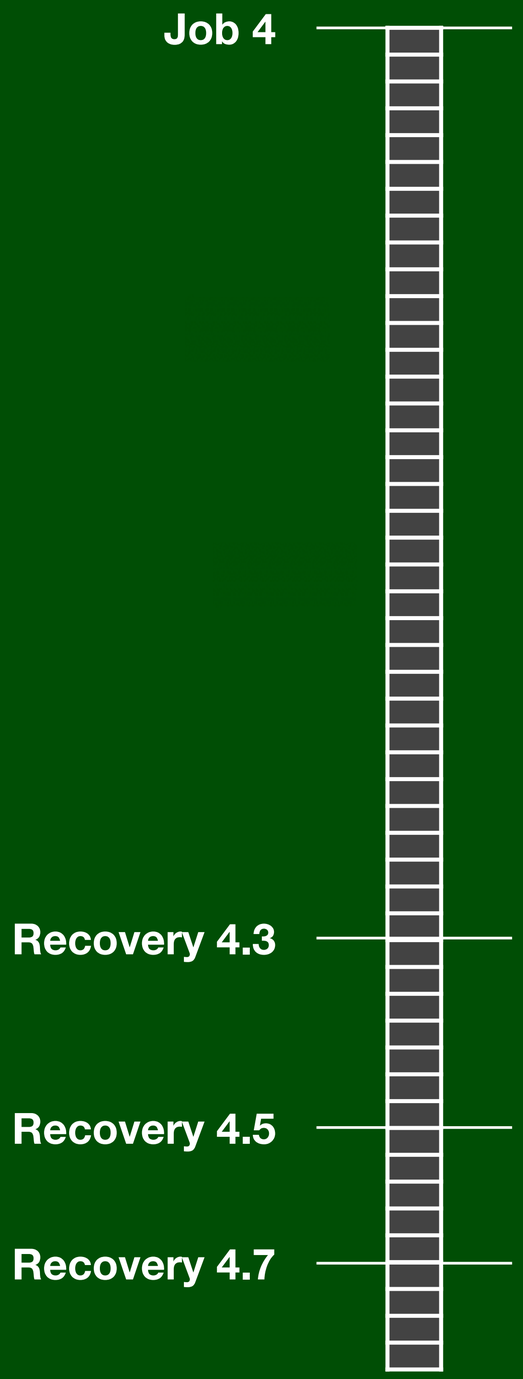
Interrupt Watchdog

JMP Watchdog

NEWJOB Watchdog

Parity

RESET



Program Abort

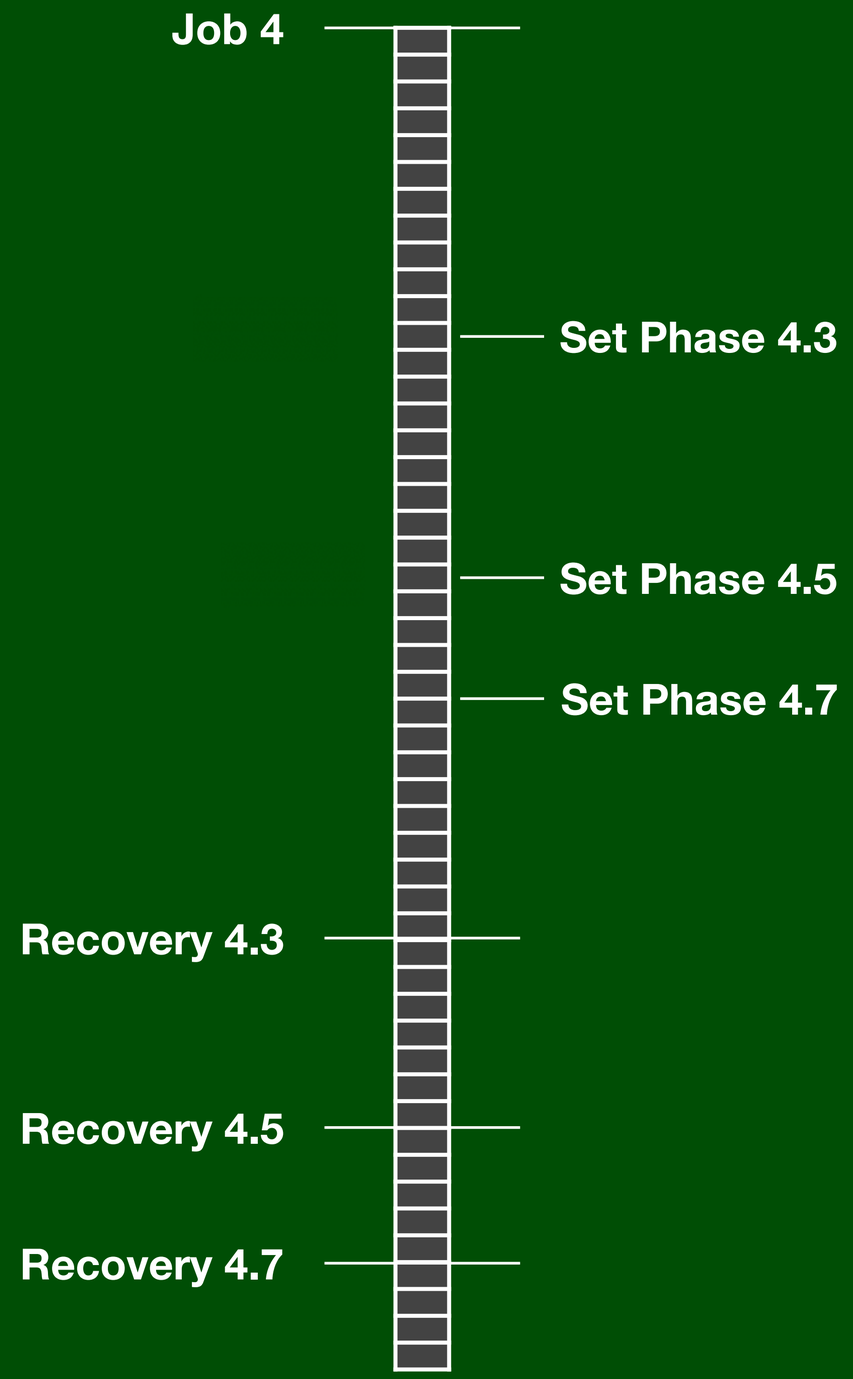
Interrupt Watchdog

JMP Watchdog

NEWJOB Watchdog

Parity

RESET



Program Abort

Interrupt Watchdog

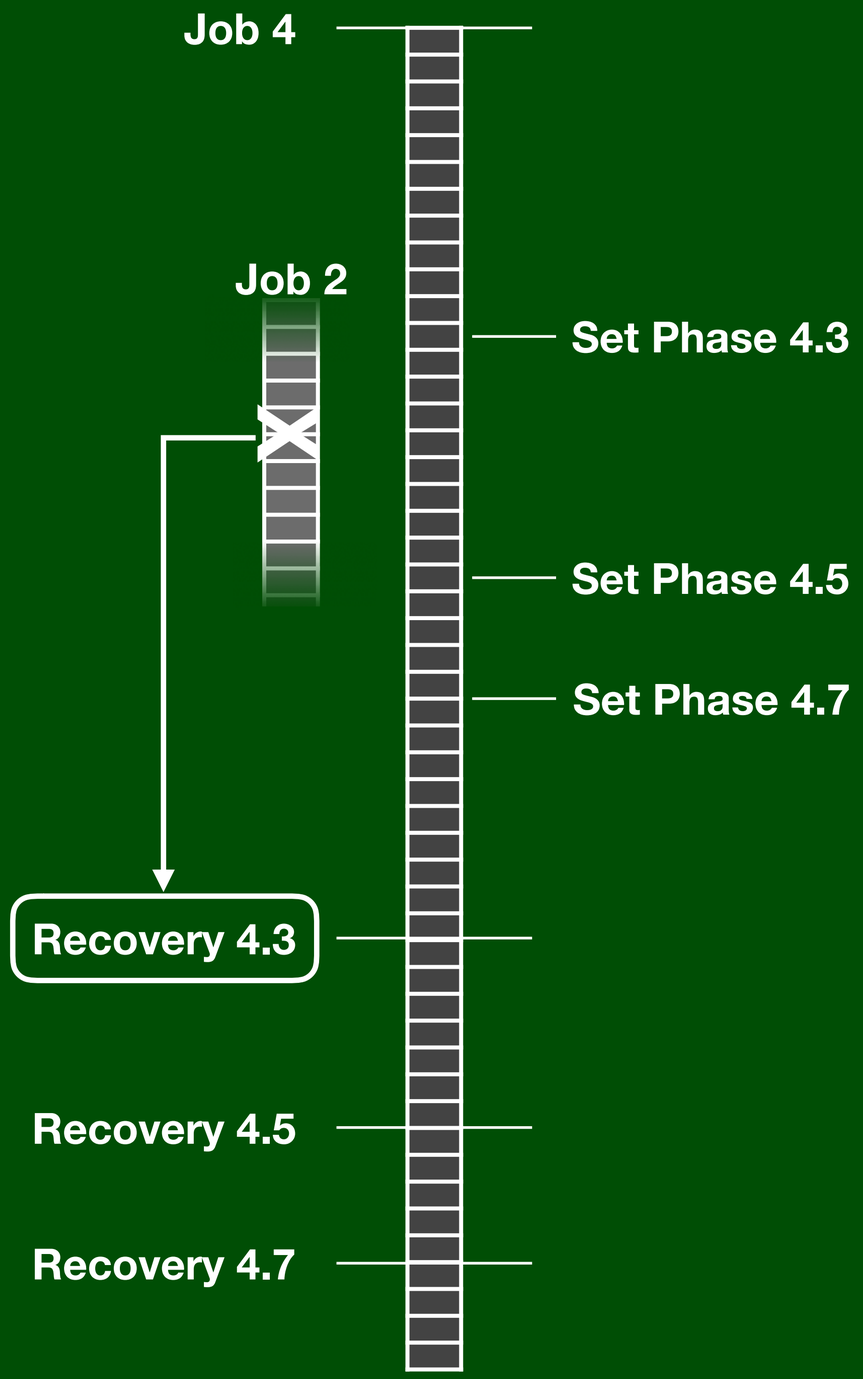
JMP Watchdog

NEWJOB Watchdog

Parity

RESET


```
call ABORT
.word 01102
```



Program Abort

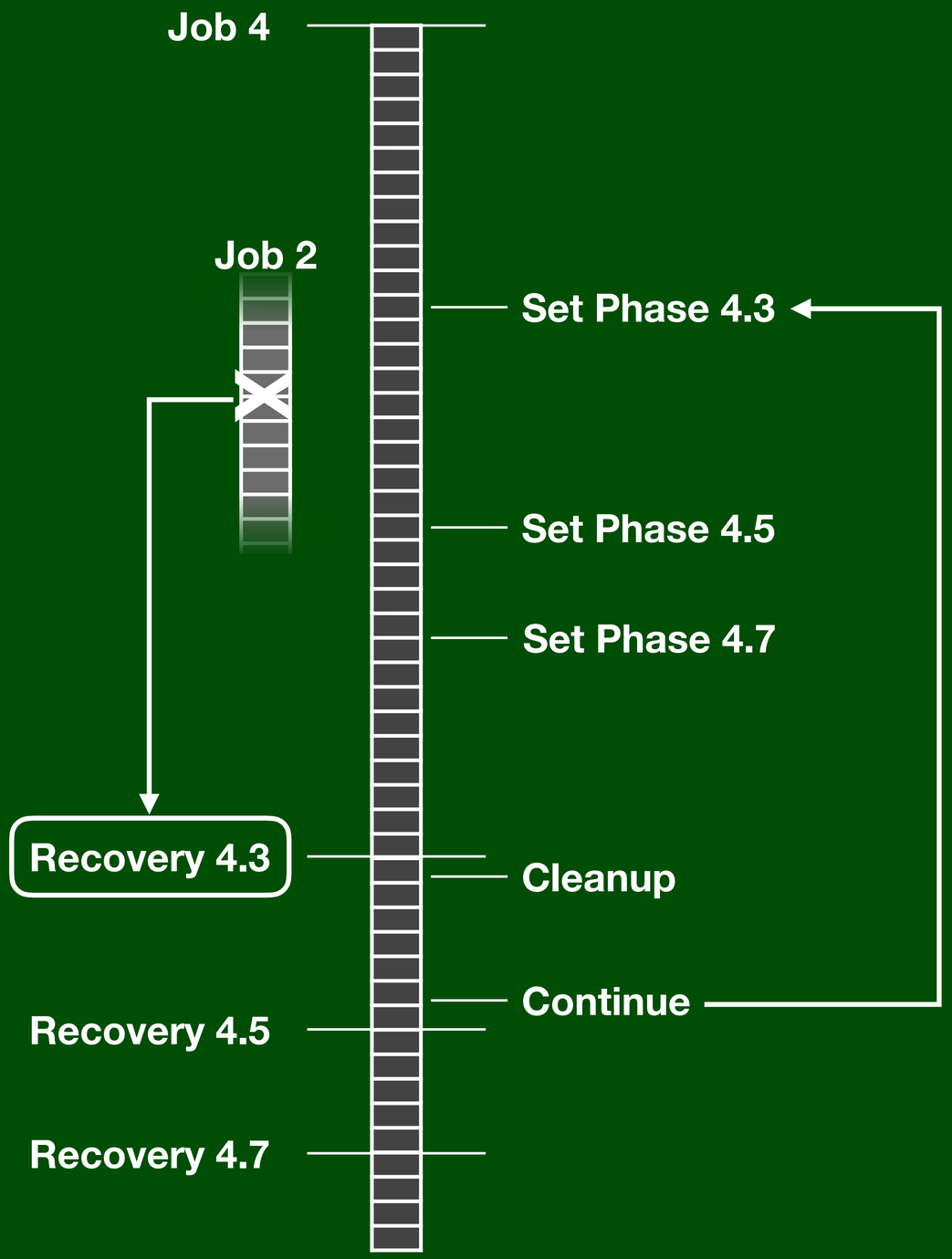
Interrupt Watchdog

JMP Watchdog

NEWJOB Watchdog

Parity

RESET



Program Abort

Interrupt Watchdog

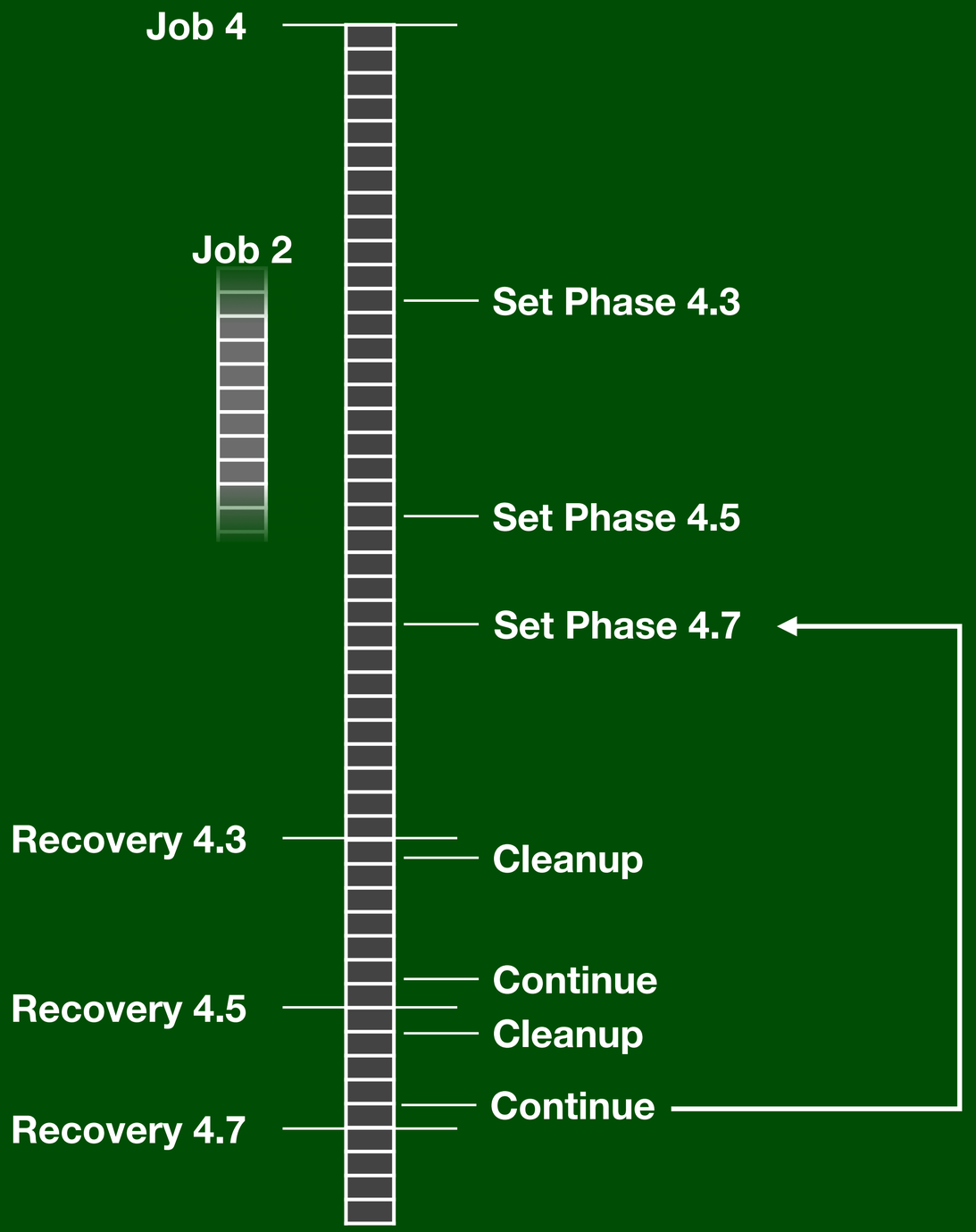
JMP Watchdog

NEWJOB Watchdog

Parity

RESET


```
call ABORT
.word 01102
```



Program Abort

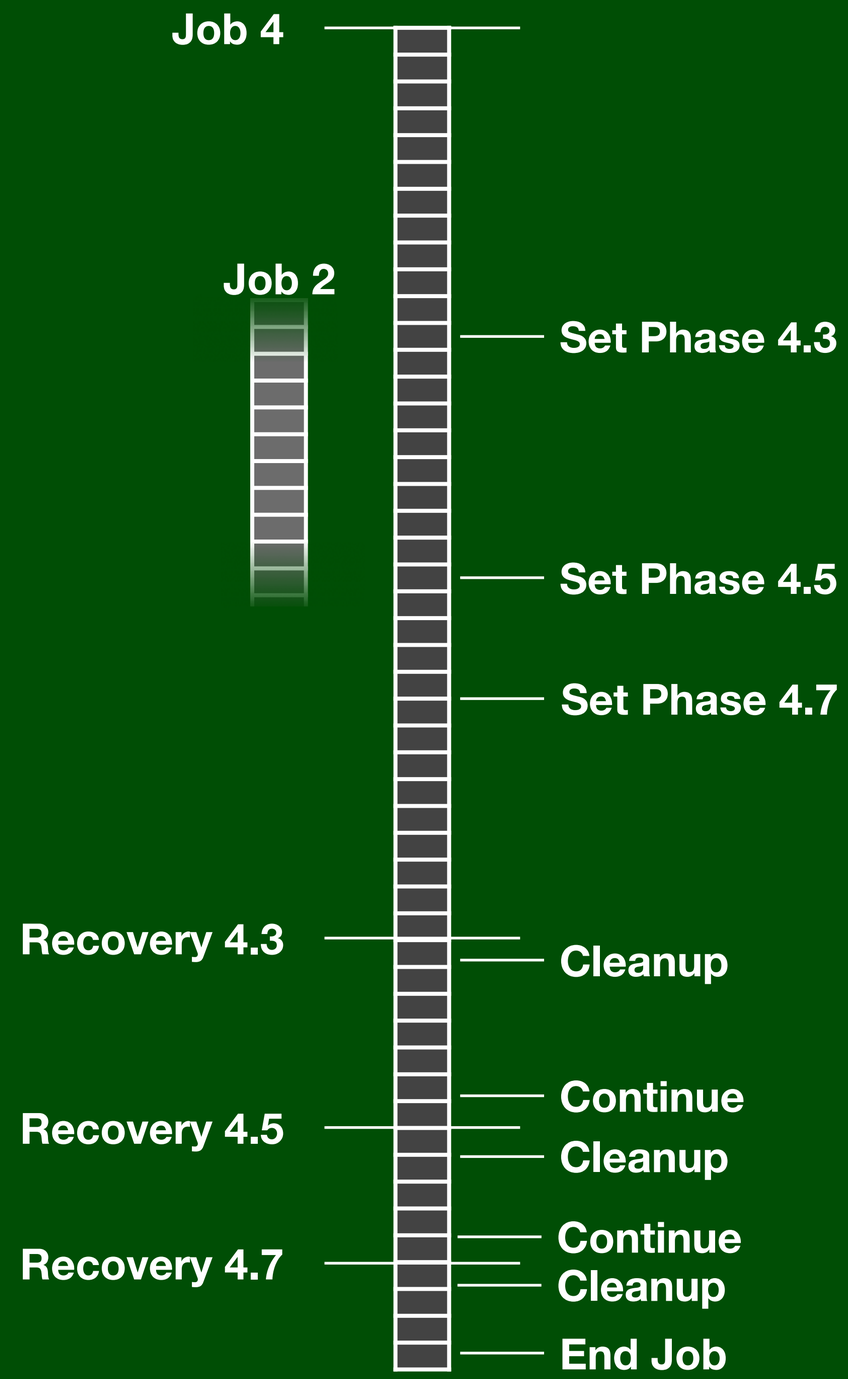
Interrupt Watchdog

JMP Watchdog

NEWJOB Watchdog

Parity

RESET



Program Abort

Program Abort

Recovery Table

Job	
1	
2	
3	
4	
5	
6	

Program Abort


```
call PHASCHNG
.word 00054
```

Phase Job

Recovery Table

Job	
1	
2	
3	
4	0005
5	
6	

Program Abort

0B77	41F
002C	420

```
call PHASCHNG
.word 00054
```

Phase Job

Recovery Table

Job	
1	
2	
3	
4	0005
5	
6	

Task 4.3

Time
PC
BB

Program Abort

0B77	41F
002C	420

```
call PHASCHNG
.word 00054
```

Phase Job

Recovery Table

Job	
1	
2	
3	
4	0005
5	
6	

Task Job

		Time
4.3		PC
		BB
		Priority
4.5		PC
		BB

Program Abort

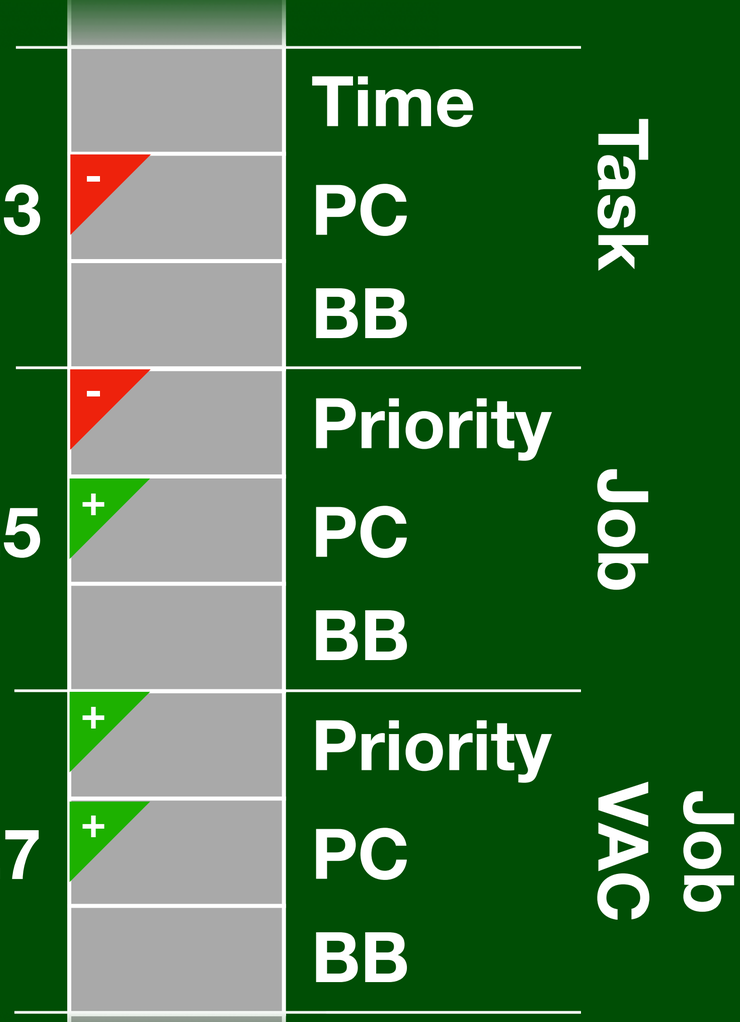
0B77	41F
002C	420

```
call PHASCHNG
.word 00054
```

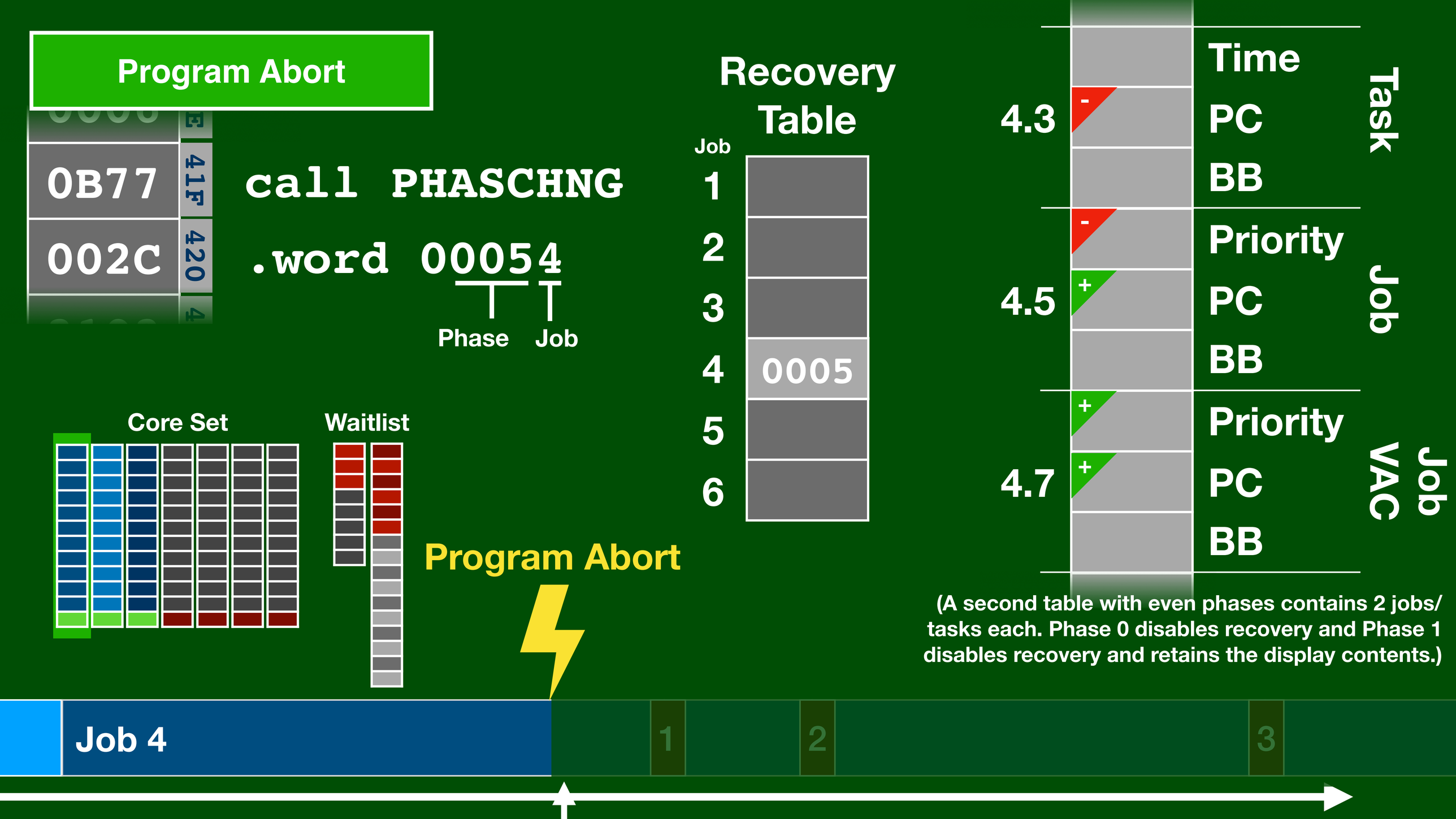
Phase Job

Recovery Table

Job	
1	
2	
3	
4	0005
5	
6	



(A second table with even phases contains 2 jobs/ tasks each. Phase 0 disables recovery and Phase 1 disables recovery and retains the display contents.)



(A second table with even phases contains 2 jobs/ tasks each. Phase 0 disables recovery and Phase 1 disables recovery and retains the display contents.)

Program Abort

0B77 41F

002C 420

call PHASCHNG

.word 00054

Phase Job

Recovery Table

Job	
1	
2	
3	
4	0005
5	
6	

Task Job VAC

4.3	-	Time
		PC
		BB
4.5	-	Priority
	+	PC
		BB
4.7	+	Priority
	+	PC
		BB

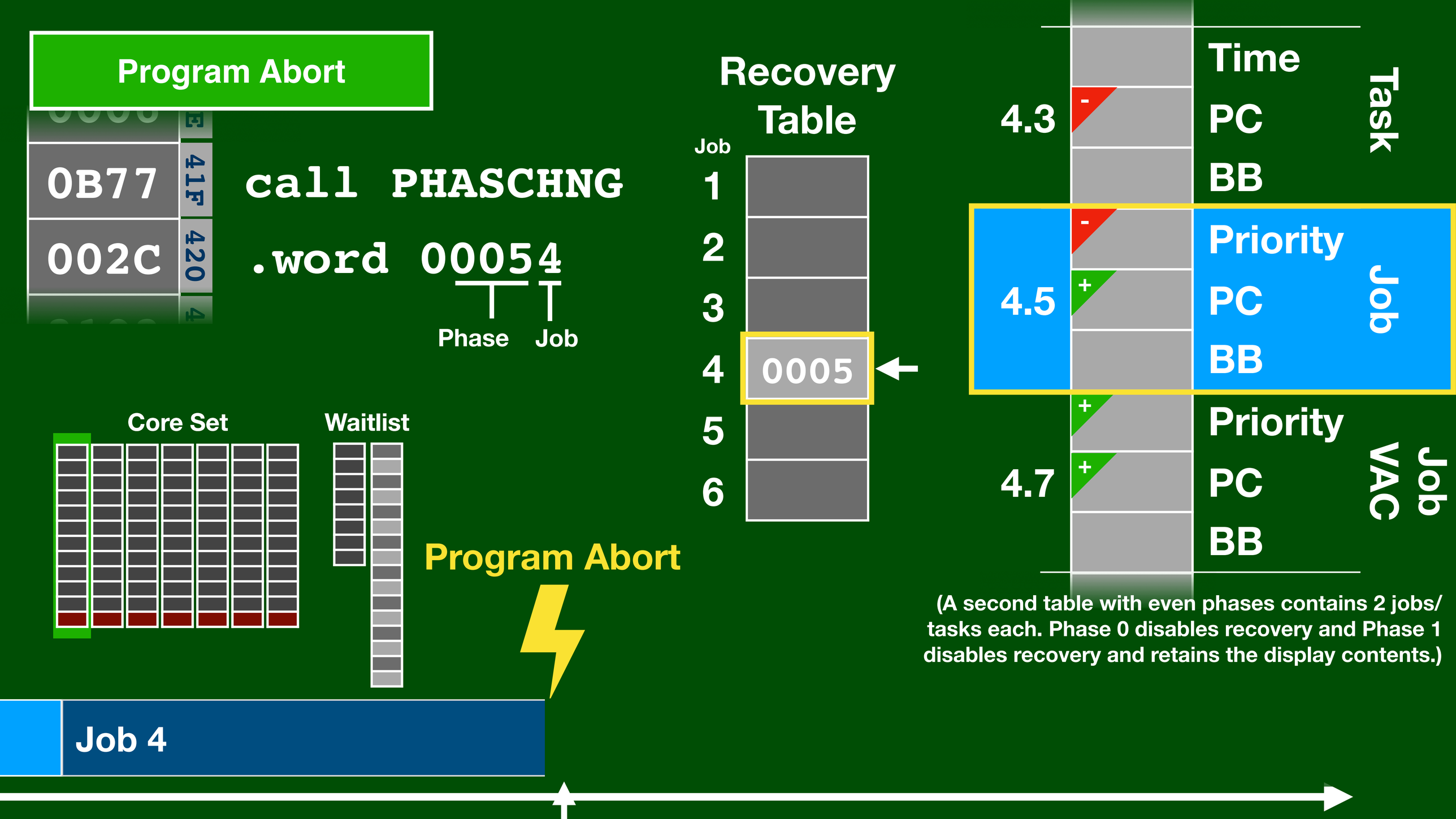
(A second table with even phases contains 2 jobs/ tasks each. Phase 0 disables recovery and Phase 1 disables recovery and retains the display contents.)

Core Set

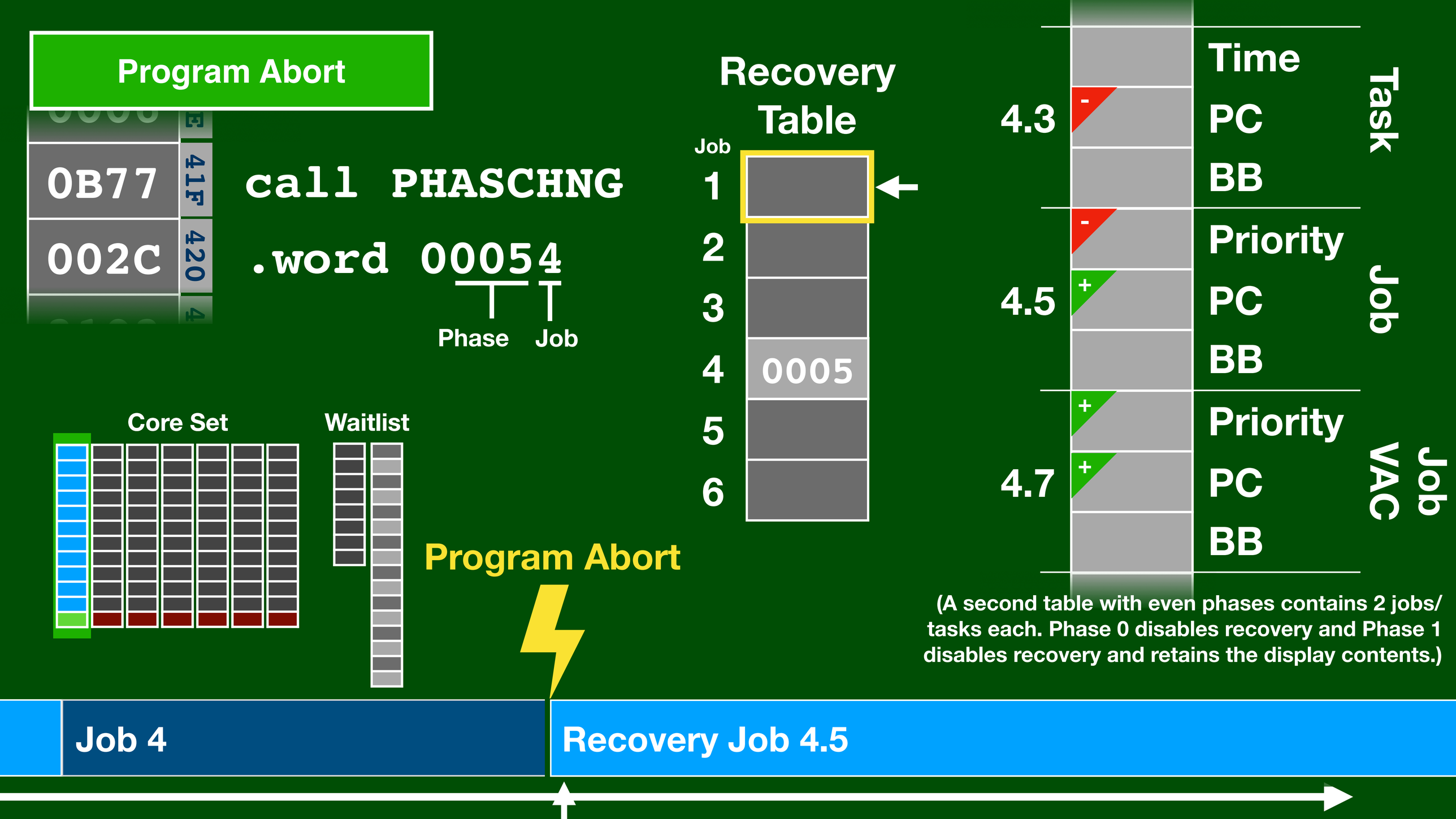
Waitlist

Program Abort

Job 4



(A second table with even phases contains 2 jobs/ tasks each. Phase 0 disables recovery and Phase 1 disables recovery and retains the display contents.)


Program Abort
0B77
41F
002C
420
call PHASCHNG
.word 00054
Phase
Job
Recovery Table
Job
1
2
3
4
0005
5
6
Task
Time
PC
BB
4.3
Job
Priority
PC
BB
4.5
Job VAC
Priority
PC
BB
4.7
Core Set
Waitlist
Program Abort
Job 4
Recovery Job 4.5
(A second table with even phases contains 2 jobs/ tasks each. Phase 0 disables recovery and Phase 1 disables recovery and retains the display contents.)

System Software

- Multitasking
- Shell
- Interpreter
- Fault Recovery
- Device Drivers
- Waitlist

System Software

Apollo Guidance Computer

Architecture

Mission Software

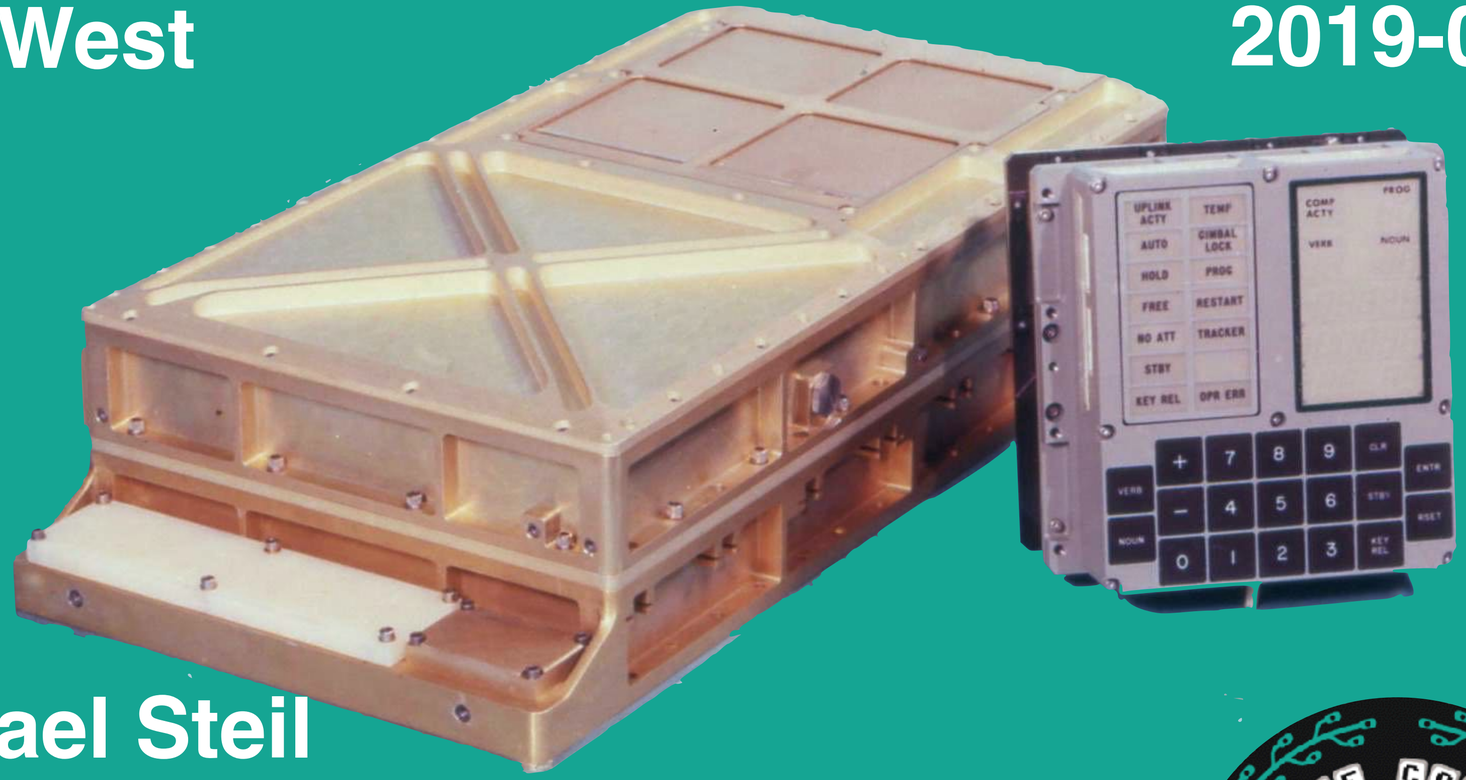
Hardware

Peripherals

System Software

VCF West

2019-08-04



Michael Steil

The Apollo Guidance Computer

@pagetable
http://www.pagetable.com/

VCF West

2019-08-04

Michael Steil
@pagetable

Christian Hessmann
@hessi

Dominik Wagner
@monkeydom

Anna-Lena Baecker
@AnnLe__

Lisa Brodner
@ellduin

Ange Albertini
http://github.com/corkami/pics

Gianluca Guida
http://www.tlbflush.org

Sven Oliver Moll
https://xayax.net

Mike Stewart
mikestewart.hcoop.net

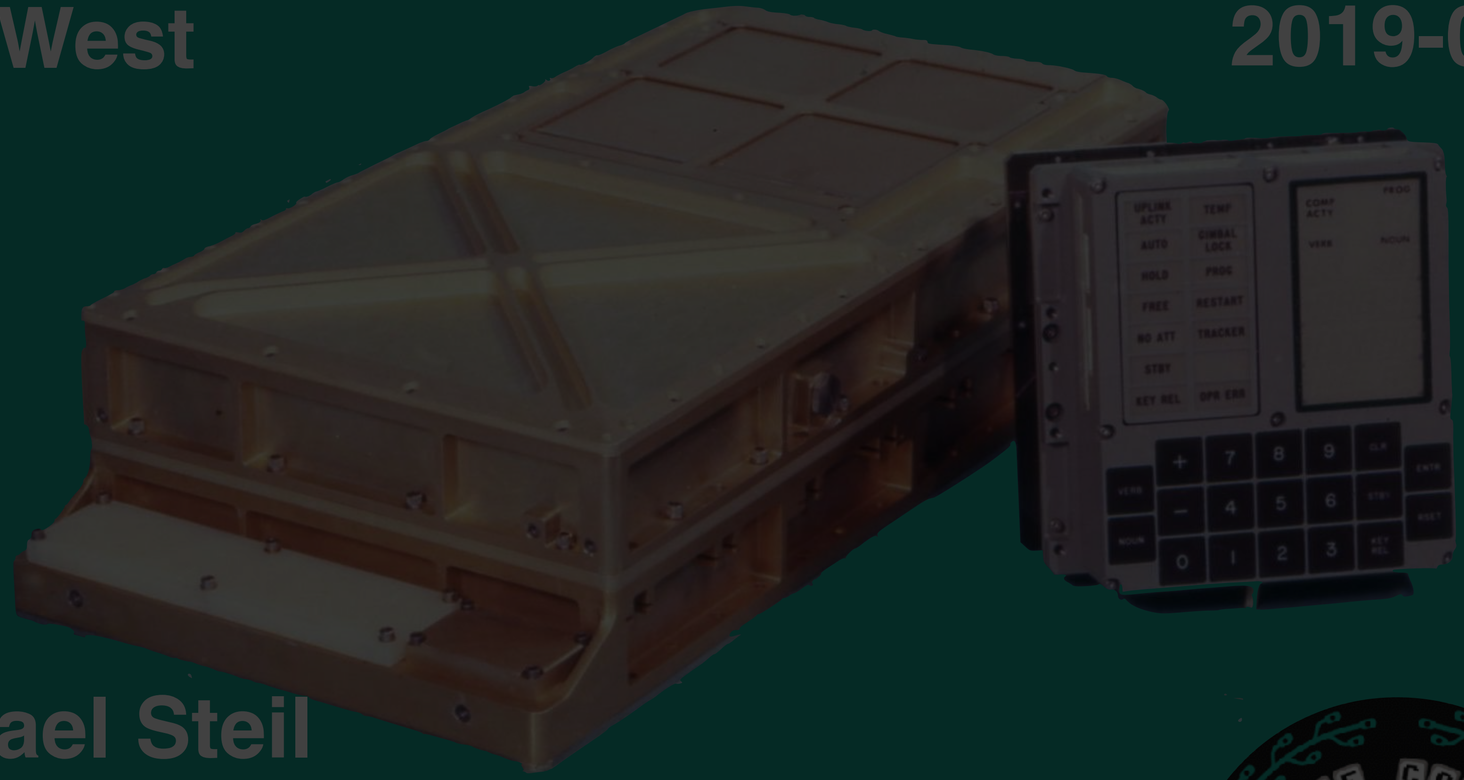
Frank O'Brien
https://www.hq.nasa.gov/alsj/franko.html

Computer

@pagetable
http://www.pagetable.com/

VCF West

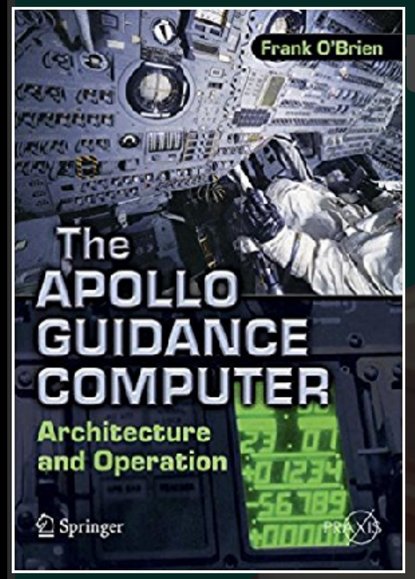
2019-08-04



Michael Steil

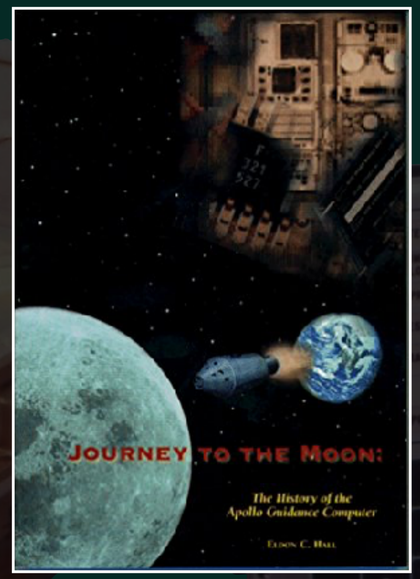
The Apollo Guidance Computer

@pagetable
http://www.pagetable.com/

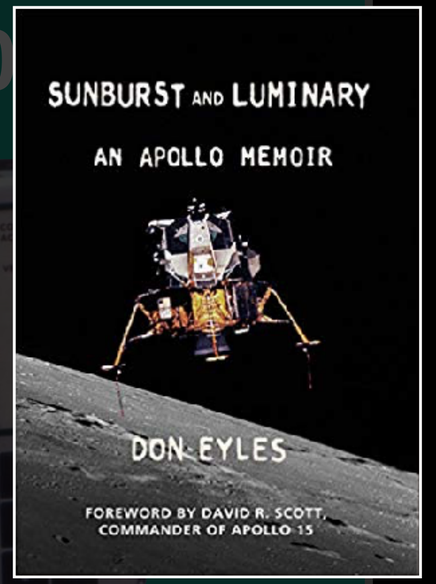


Frank O'Brien
The Apollo Guidance Computer

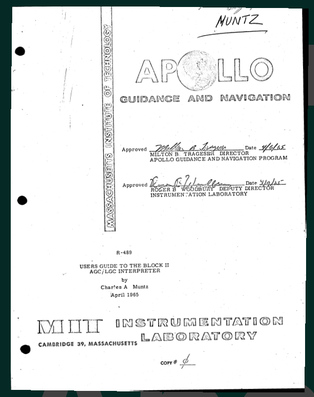
David A. Mindell
Digital Apollo



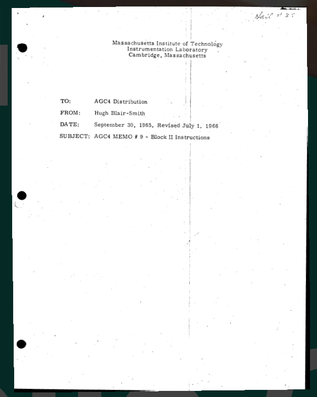
Eldon C. Hall
Journey to the Moon



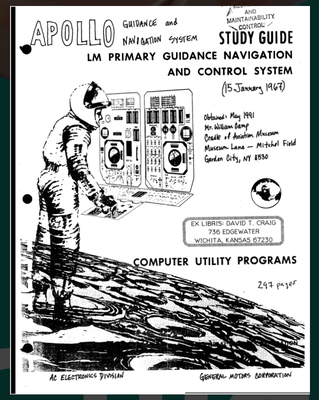
Don Eyles
Sunburst and Luminary



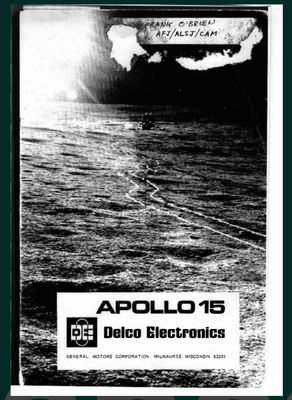
1687.pdf



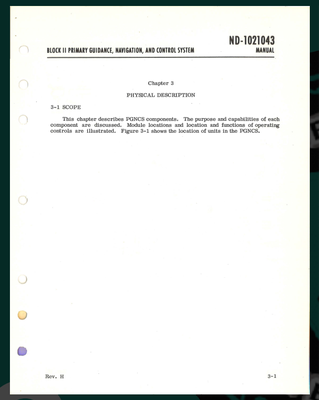
1689.pdf



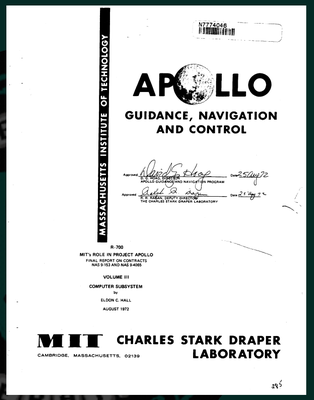
1709.pdf



A15Delco.pdf



HSI-208435-002.pdf



mitsrolevoliii.pdf

Home (AGC)
AGS
LVDC
Gemini
Downloads
Document library
Change log
Bug and issues
Developer info
Code repository
Archival scans

Virtual AGC — AGS — LVDC — Gemini

HOME PAGE
Project Overview

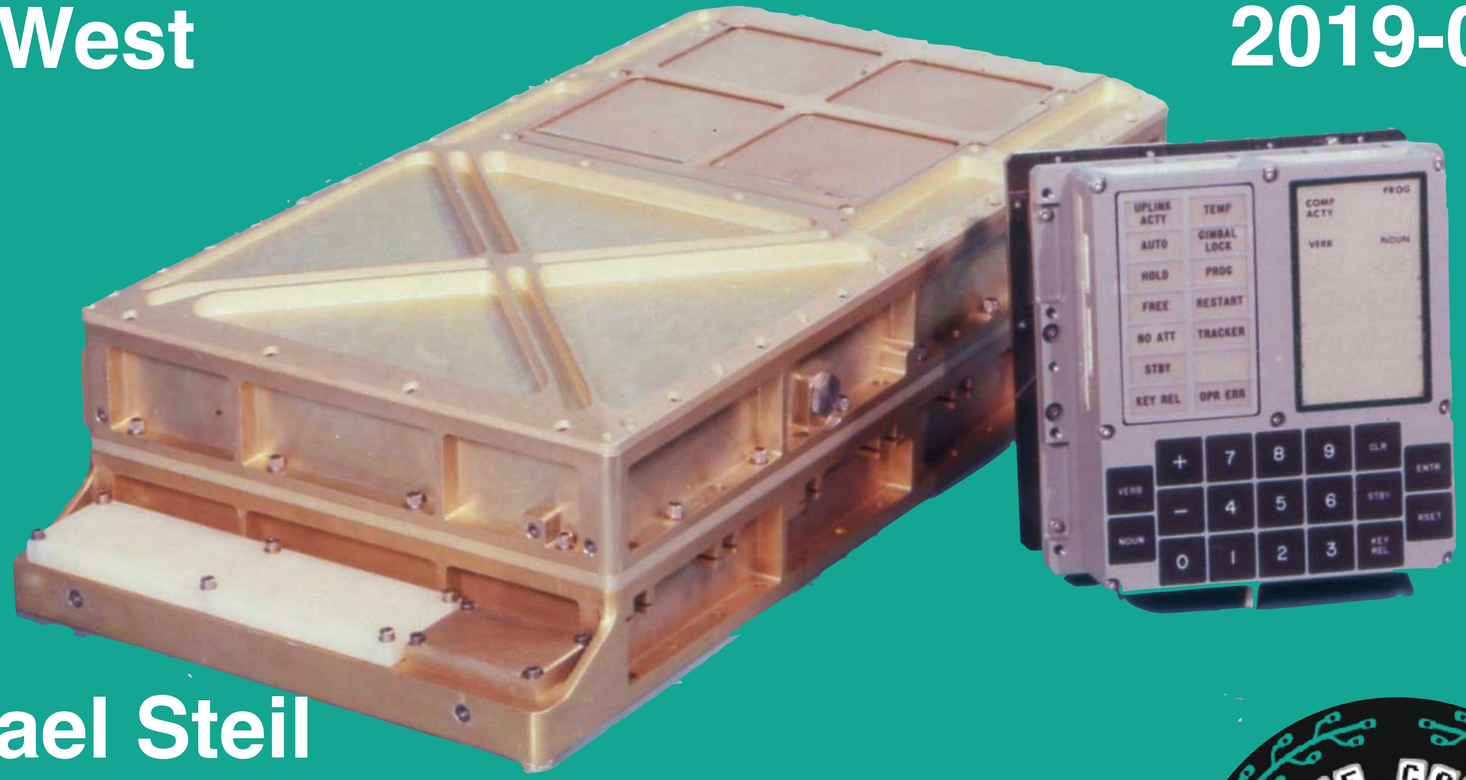
https://www.ibiblio.org/apollo/

FAQ
yaAGC
yaYUL
yaDSKY
yaOtherStuff
Luminary
Colossus
Block 1 AGC
Block 2 AGC Language
Physical Implementations
Do It Yourself

@pagetable
pagetable.com/

VCF West

2019-08-04



Michael Steil

The Apollo Guidance Computer

@pagetable
http://www.pagetable.com/