COMPUTER SYSTEMS ORGANIZATION

Processors
Primary Memory
Secondary Memory
Input/Output
Microarchitecture level

Last lecture recap

- Imbalance Between CPU and Memory
  - Cache memory (L1, L2, and L3)
  - Locality Principle and cache line

- Cache Memory Mapping Techniques
  - Direct mapping
  - Associative mapping
  - Set - Associative mapping

- Cache Replacement Algorithms
  - Least Recently Used (LRU)
  - First In First Out (FIFO)
  - Least Frequently Used (LFU)
  - Random
  - Clock algorithm

- Memory Packaging and Types
  - Single Inline Memory Module (SIMM)
  - Dual Inline Memory Module(DIMM)
  - DDR3 DIMMS
  - Small Outline DIMM (SO-DIMM)
- Magnetic Disks
- Integrated Drive Electronic (IDE)
- Small Computer System Interface (SCSI)

SECONDARY MEMORY

Redundant Array of Inexpensive Disks (**RAID**)

Redundant Array of Inexpensive Disks (**RAID**)

≡

SECONDARY MEMORY

Redundant Array of Inexpensive Disks (**RAID**)

RAID vs. SLED

- Redundant Array of Inexpensive Disks
- Single Large Expensive Disk

Idea behind RAID

- Install a box full of disks next to the computer, typically a large server, replace the disk controller card with a RAID controller, copy the data over to RAID
- RAID should look like a SLED to the operating system but have better performance and reliability
- No software changes are required
- Data are distributed over the drives to allow parallel operations

Since **SCSI** disks have good performance, low price, and the ability to have up to 7 drives on a single controller (15 for wide SCSI), it is natural that many RAIDs consist of a RAID SCSI controller plus a box of SCSI disks that appear to the operating system as a single large disk.

Striping: distributing data over multiple drives

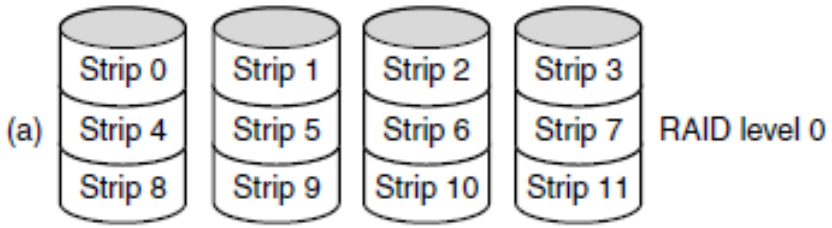
1 strip = k sectors

sectors 0 to k-1 as strip 0
sectors k to 2k – 1 as strip 1
sectors 2k to 3k – 1 as strip 2
. . . . . .

e.g. k = 2
sectors 0 to 1 as strip 0
sectors 2 to 3 as strip 1
sectors 4 to 5 as strip 2
. . . . . .

SECONDARY MEMORY

Redundant Array of Inexpensive Disks (**RAID**)



RAID 0

- Works best with large requests, the bigger the better

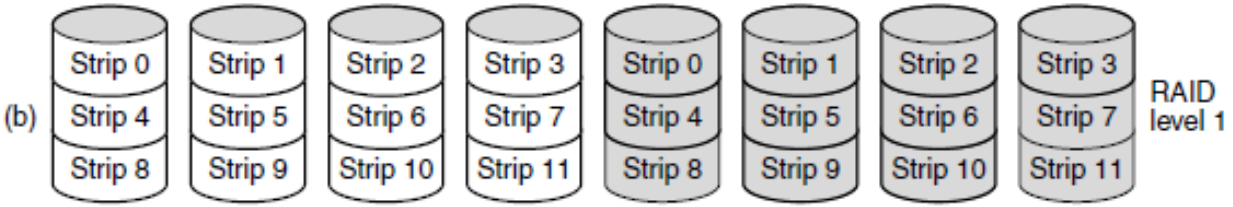
- Works worst with operating systems that habitually ask for data one sector at a time. The results will be correct, but there is no parallelism and hence no performance gain

- No redundancy, not a really RAID

- Work with strips of sectors

SECONDARY MEMORY

Redundant Array of Inexpensive Disks (**RAID**)



Shaded are backup and parity disks

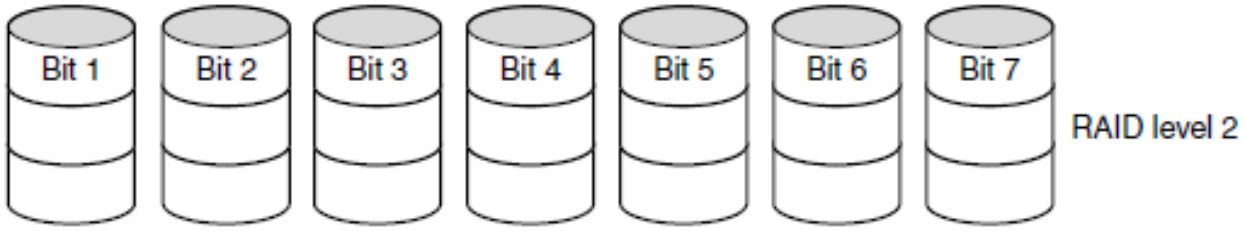
RAID 1

- Duplicate all disks
- Work with strips of sectors,
- Half primary disks and half backup disks
- True RAID
- On a write, every strip is written twice. On a read, either copy can be used, distributing the load over more drives. Consequently, write performance is no better than for a single drive, but read performance can be up to twice as good.

- Fault tolerance is excellent: if a drive crashes, the copy is simply used instead.

- Recovery consists of simply installing a new drive and copying the entire backup drive to it.

RAID 2



Error detections and corrections
Hamming code(m = 4, r = 3)

$$(m + r + 1) \leqslant 2^r$$

m = 4, r = 3 ⇒ 8 ≤ 8

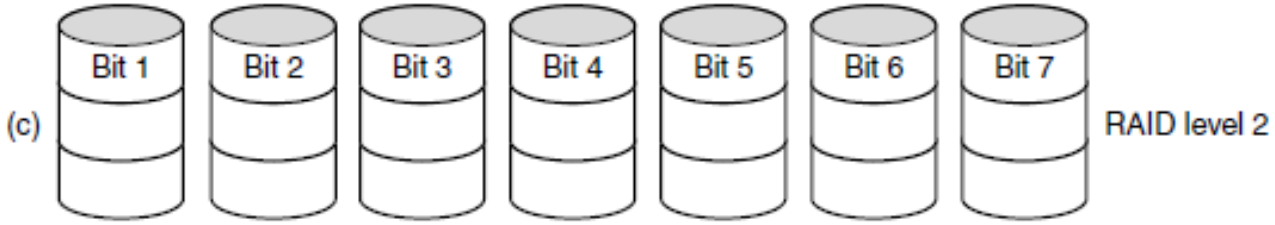
Data word = abcd
Parity bits = xyz

x = a⊕b⊕d
y = a⊕c⊕d
z = b⊕c⊕d

Code word	**x**	**y**	**a**	**z**	**b**	**c**	**d**
Position	1	2	3	4	5	6	7
	2^0	2^1	3	2^2	5	6	7

SECONDARY MEMORY

Redundant Array of Inexpensive Disks (**RAID**)



Shaded are backup and parity disks

RAID 2

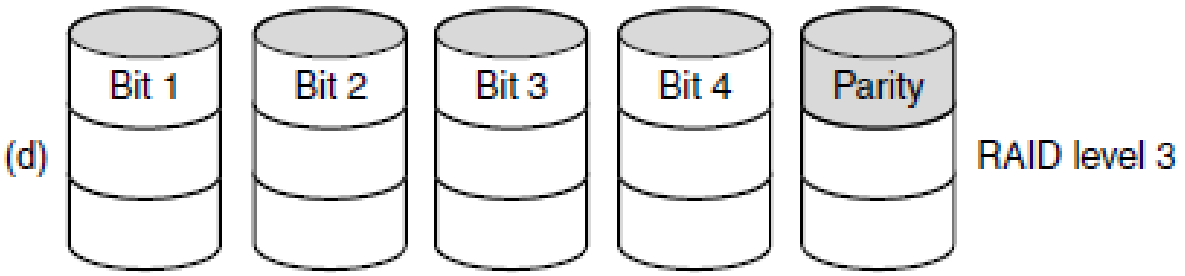
- On a word basis or a byte basis
- 7-bit hamming code over 7 drives, 1 bit per drive

- Exactly synchronized in terms of arm position and rotational position. It also asks a lot of the controller, since it must do a Hamming checksum every bit time.

- Losing one drive did not cause problems, because loss of a drive amounted to losing 1 bit in each 7-bit word read, something the Hamming code could handle on the fly.

SECONDARY MEMORY

Redundant Array of Inexpensive Disks (**RAID**)



Shaded are backup and parity disks

RAID 3

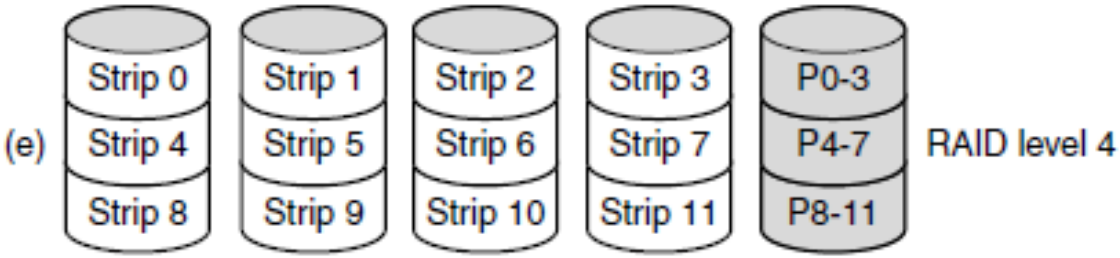
- Simplified version of RAID level 2

- A single parity bit is computed for each data word and written to a parity drive

- The drives must be exactly synchronized, since individual data words are spread over multiple drives.

SECONDARY MEMORY

Redundant Array of Inexpensive Disks (**RAID**)



Shaded are backup and parity disks

RAID 4

- Work with strips again

- Not individual word with parity

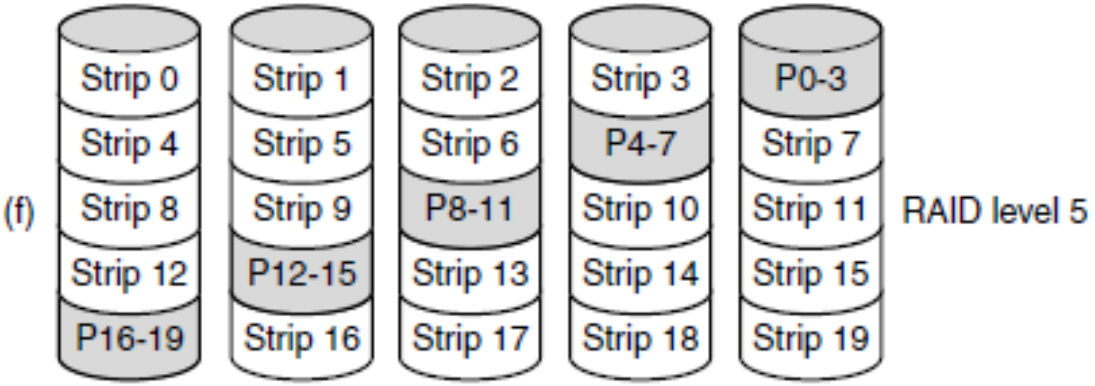
- Strip-to-Strip parity written onto the extra drive

- Not require synchronized drives

- If one sector is changed, all the drives must be read in order to recalculate the parity, which then must be rewritten

SECONDARY MEMORY

Redundant Array of Inexpensive Disks (**RAID**)



Shaded are backup and parity disks

RAID 5

- Distributed the parity bits uniformly over all drives
- Eliminating the heavy load on the parity drive
- Recovery is a complex in a drive crash
- Works with strips again, not individual words with parity, and do not require synchronized drives

SECONDARY MEMORY

Solid-State Disks(SSD)

Disks made from nonvolatile flash memory

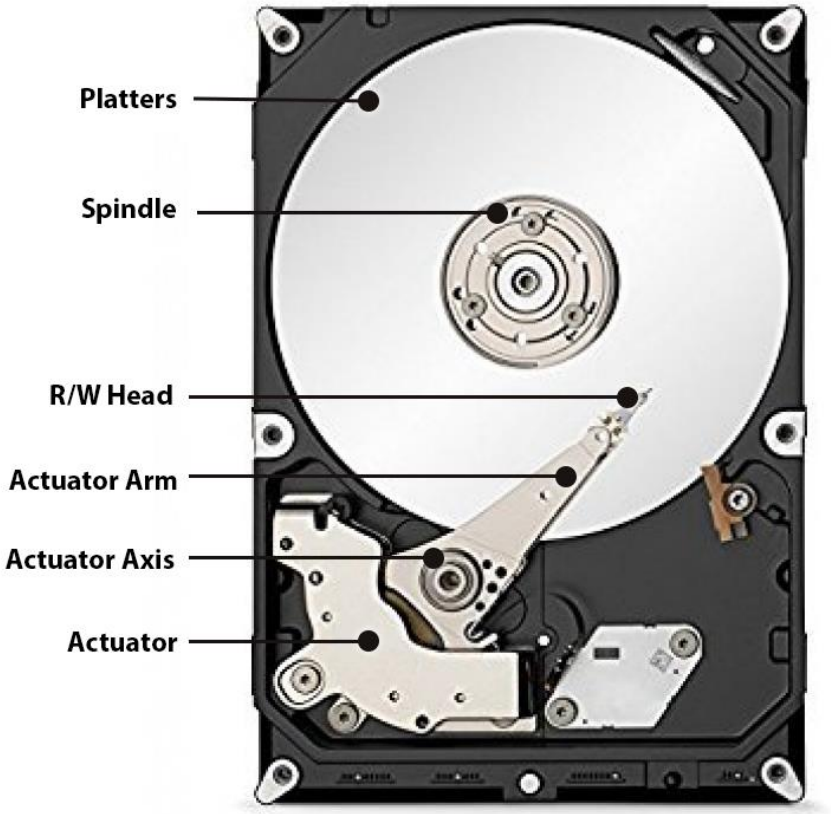
Advantages:
Because SSDs are essentially memory, they have superior performance to spinning disks and have zero seek time. While a typical magnetic disk can access data up to 100 MB/sec,

A SSD can operate two to three times faster and because the device has no moving parts, it is particularly suited for use in notebook computers

Disadvantages:
Compared to magnetic disks, is their cost. While magnetic disks cost pennies/gigabyte, a typical SSD will cost one to three dollars/gigabyte,

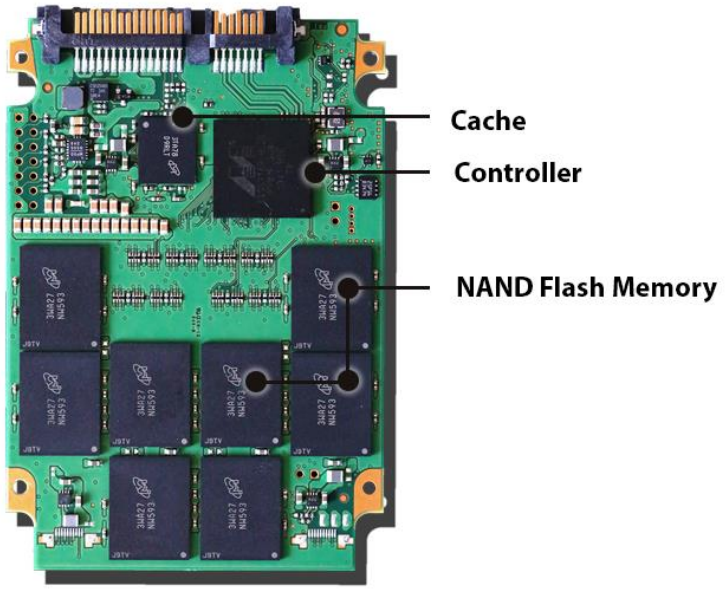
Failure rate. A typical flash cell can be written only about 100,000 times before it will no longer function.

HDD
3.5″



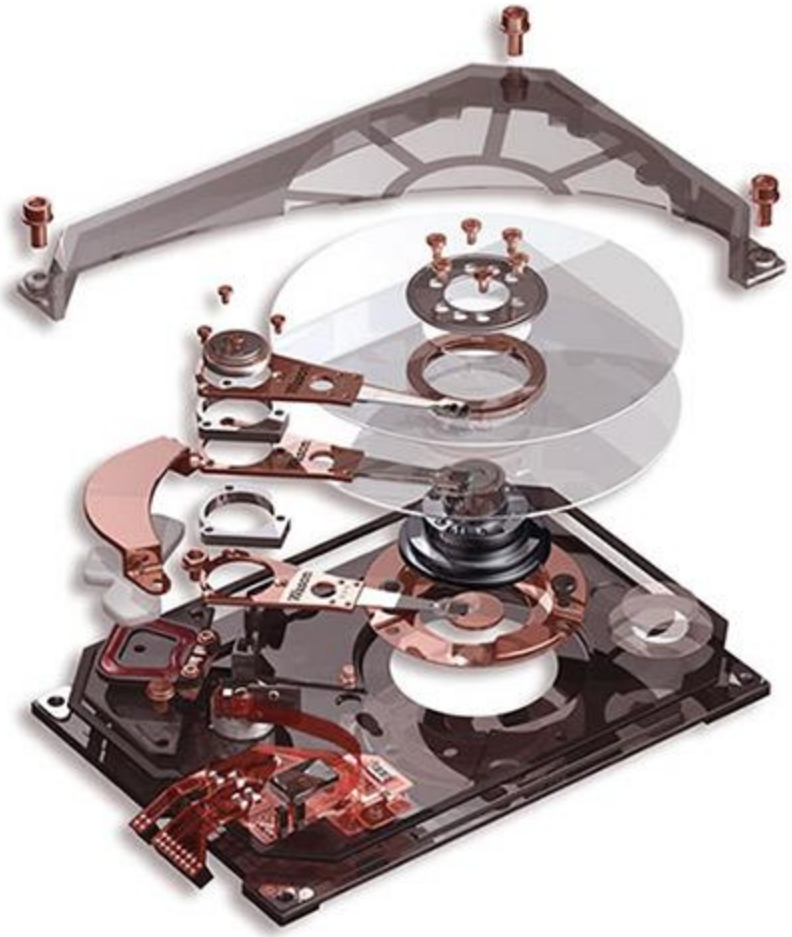
Shock resistant up to 55g (operating)
Shock resistant up to 350g (non-operating)

SSD
2.5″

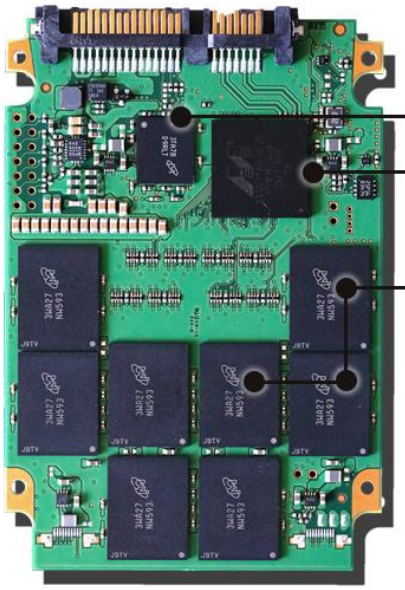


Shock resistant up to 1500g
(operating and non-operating)

https://www.backblaze.com/blog/ssd-vs-hdd-future-of-storage/



SSD
2.5″



Shock resistant up to 1500g
(operating and non-operating)

https://www.backblaze.com/blog/ssd-vs-hdd-future-of-storage/

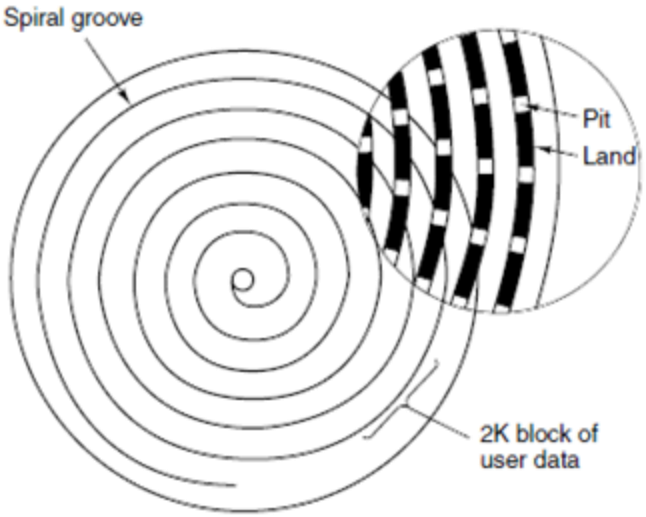
SECONDARY MEMORY

Compact Disc (CD)

CD Preparation

- Use a high power infrared laser to burn 0.8-micron diameter holes in a coated glass master disk
- **Pit:** Depression in the polycarbonate substrate
- **Land:** unburned areas
- Pit/land or land/pit transition is 1 and otherwise 0

- **Rotation Rate**
- Constant linear velocity, which different from magnetic disk with constant angular velocity



SECONDARY MEMORY

CD-Rewritables, DVD, and Blu-ray

CD-Rewritables:

A. Same size media as CD-R
B. More expensive than the CD-R blanks
C. Rewriteable

Digital Video Disk (DVD)

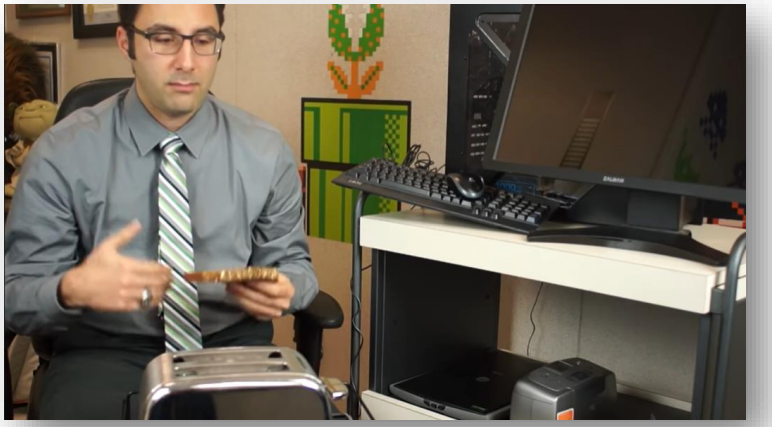
A. Single-sided, single-layer (4.7 GB)
B. Single-sided, dual-layer (8.5 GB)
C. Double-sided, single-layer (9.4 GB)
D. Double-sided, dual-layer (17 GB)

Blu-ray

A. Single-sided 25 GB
B. Double-sided ones 50 GB

Processors
Primary Memory
Secondary Memory
Input/Output
Microarchitecture level

Computer Hardware & Software Lesson Part 1



https://www.youtube.com/watch?v=8UyJMiYqvs4

Computer Hardware & Software Lesson Part 2

https://www.youtube.com/watch?v=gaN1SKti3ts

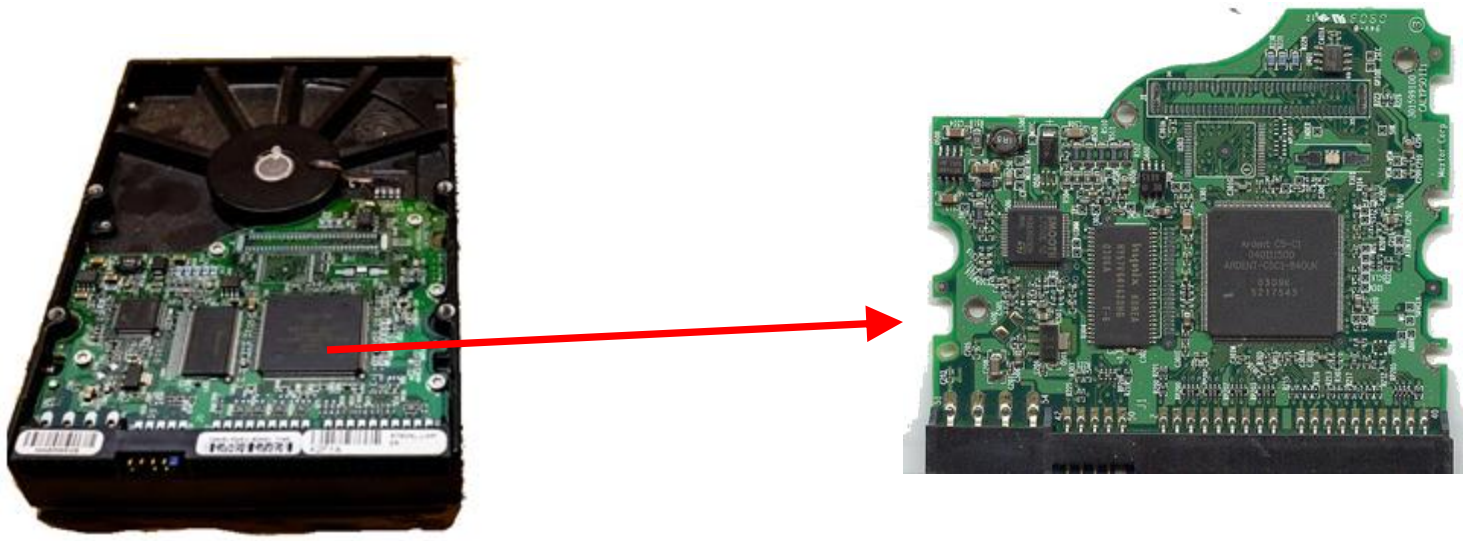
INPUT/OUTPUT

I/O Device

Each I/O device consists of two parts:

1) Controller
2) I/O device itself

Controller is used to control its I/O device and handle bus access for it.



INPUT/OUTPUT

I/O Device

I/O devices also interact heavily with the operating system.

The software that talks to a controller, giving it commands and accepting responses, is called a **device driver.** the driver has to be put into the operating system so it can run in kernel mode.

Every controller has a small number of registers that are used to communicate with it.

For example, a minimal disk controller might have registers for specifying the disk address, memory address, sector count, and direction (read or write).

To activate the controller, the driver gets a command from the operating system, then translates it into the appropriate values to write into the device registers.

INPUT/OUTPUT

I/O Device

Input and output can be done in three different ways:

1. **Busy waiting**

User program issues a system call, which the kernel then translates into a procedure call to the appropriate driver.

The driver then starts the I/O and sits in a tight loop continuously polling the device to see if it is done (usually there is some bit that indicates that the device is still busy). When the I/O has completed, the driver puts the data (if any) where they are needed and returns.

The operating system then returns control to the caller. This method is called busy waiting and has the disadvantage of tying up the CPU polling the device until it is finished.

INPUT/OUTPUT

I/O Device

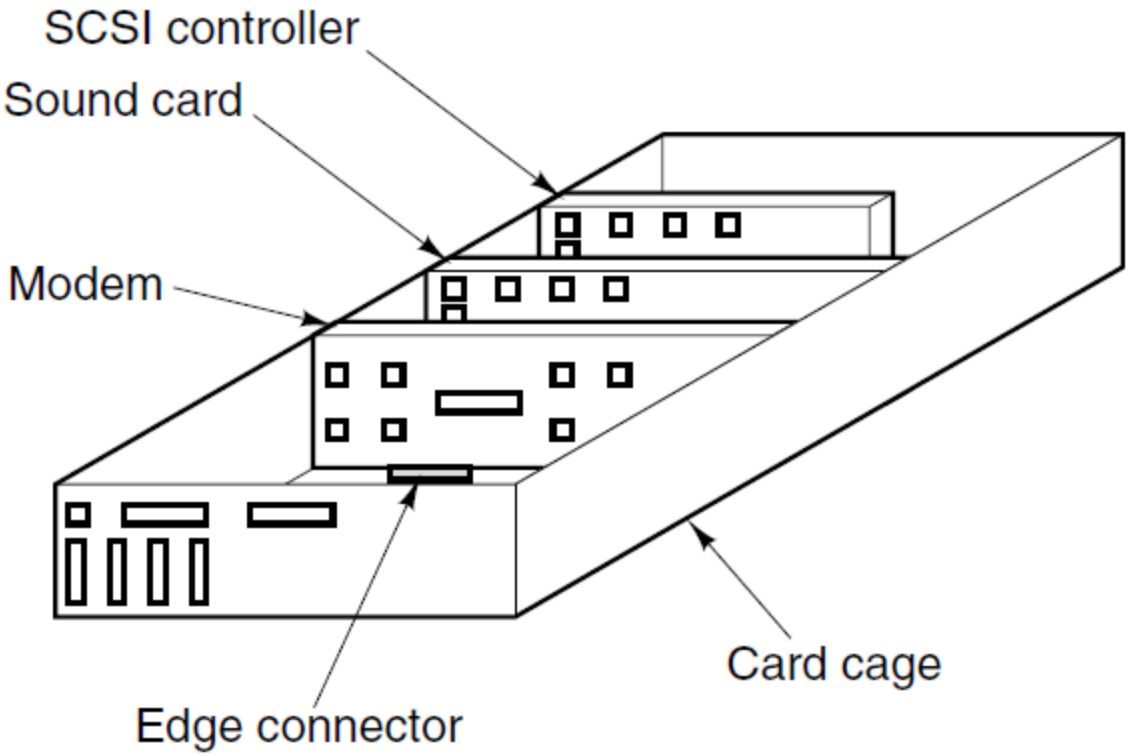
2. **Using interrupt**

The second method is for the driver to start the device and ask it to give an interrupt when it is finished. At that point the driver returns.
The operating system then blocks the caller if need be and looks for other work to do. When the controller detects the end of the transfer, it generates an **interrupt** to signal completion.

3. **Use of special hardware** e.g. DMA (**Direct Memory Access**)

The CPU sets up the DMA chip, telling it how many bytes to transfer, the device and memory addresses involved, and the direction, and lets it go. When the DMA chip is done, it causes an interrupt.

INPUT/OUTPUT



Physical structure of a personal computer.

INPUT/OUTPUT

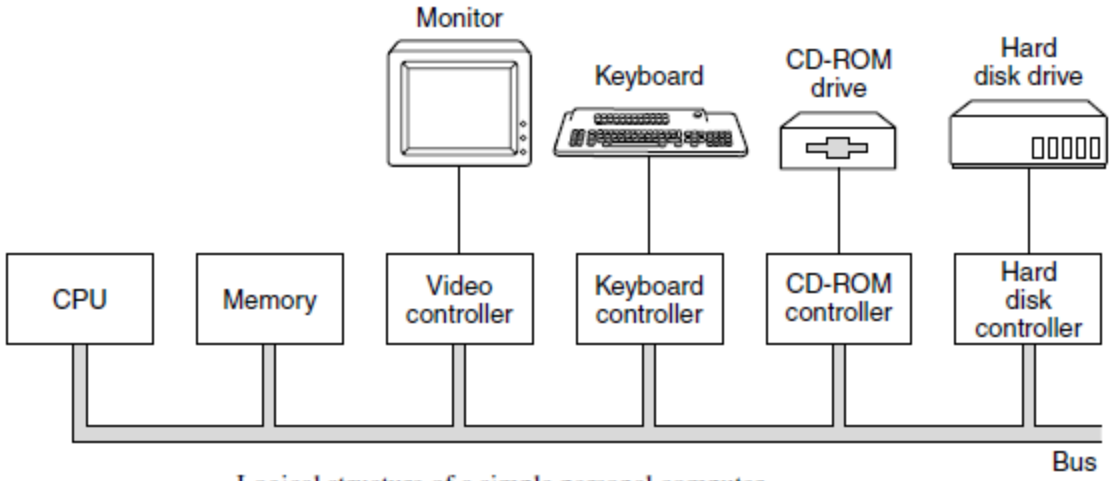
Buses

Bus

– Used by the I/O controller and the CPU for fetching instructions and data

- **Bus Arbiter**

– Decide who go first when the CPU and I/O controller want to use the bus at the same time; I/O devices are given preference over the CPU



Logical structure of a simple personal computer.

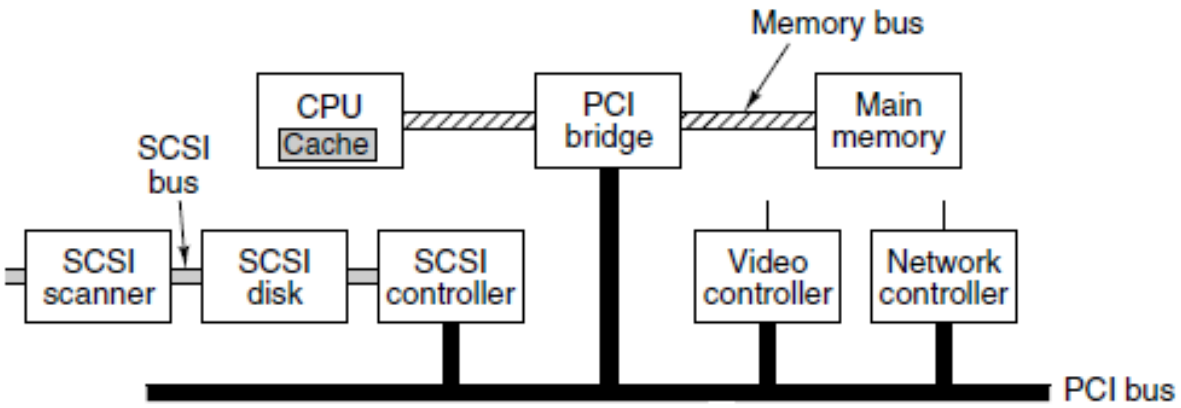
INPUT/OUTPUT

Buses

The PCI vs. ISA Buses

Industry Standard Architecture (ISA) vs. Peripheral Component Interconnect (PCI)

– CPU memory traffic does not go over the PCI bus, high bandwidth peripherals can connect to the PCI bus directly



A typical PC built around the PCI bus. The SCSI controller is a PCI device.

INPUT/OUTPUT

Keyboards

- On personal computer, when a key is depressed, an interrupt is generated and the keyboard interrupt handler is started

– The interrupt handler reads a hardware register inside the keyboard controller to get the number of the key that was just depressed

– When a key is released, a second interrupt is caused

INPUT/OUTPUT

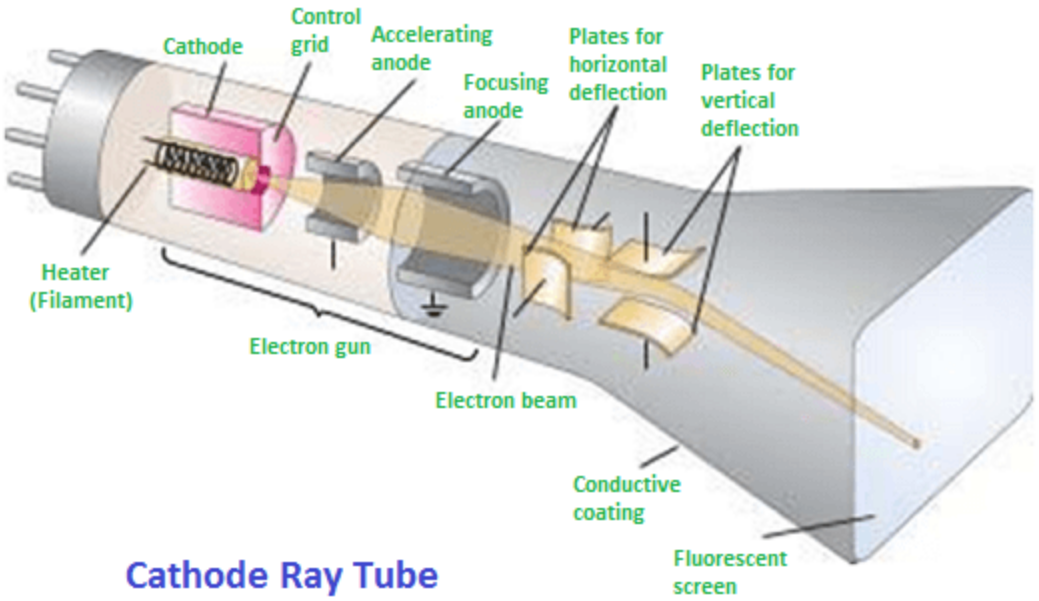
Monitors

- **CRT** (Cathode Ray Tube) monitors
- **Flat Panel Displays**: LCD (Liquid Crystal Display) monitors

Video RAM: both displays are refreshed 60- 100 times per second from a special memory --- video RAM

CRT

- A gun shoots an electron beam against a phosphorescent screen near the front of the tube
- Raster scan, a full screen is normally repainted between 30 and 60 times a Second



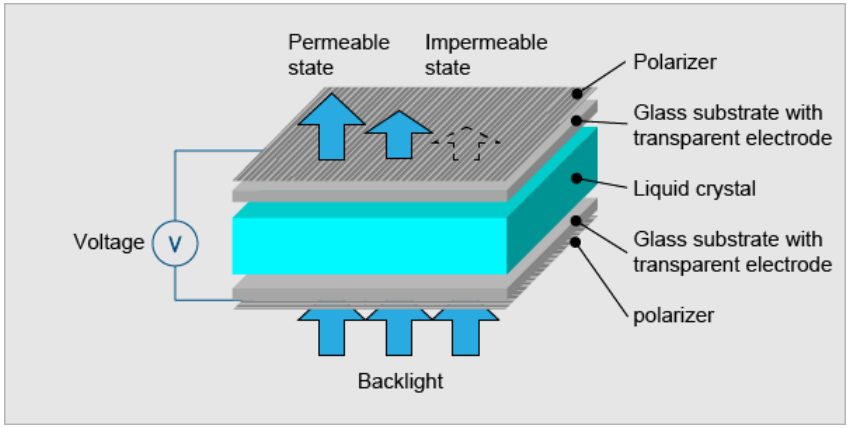
Cathode Ray Tube

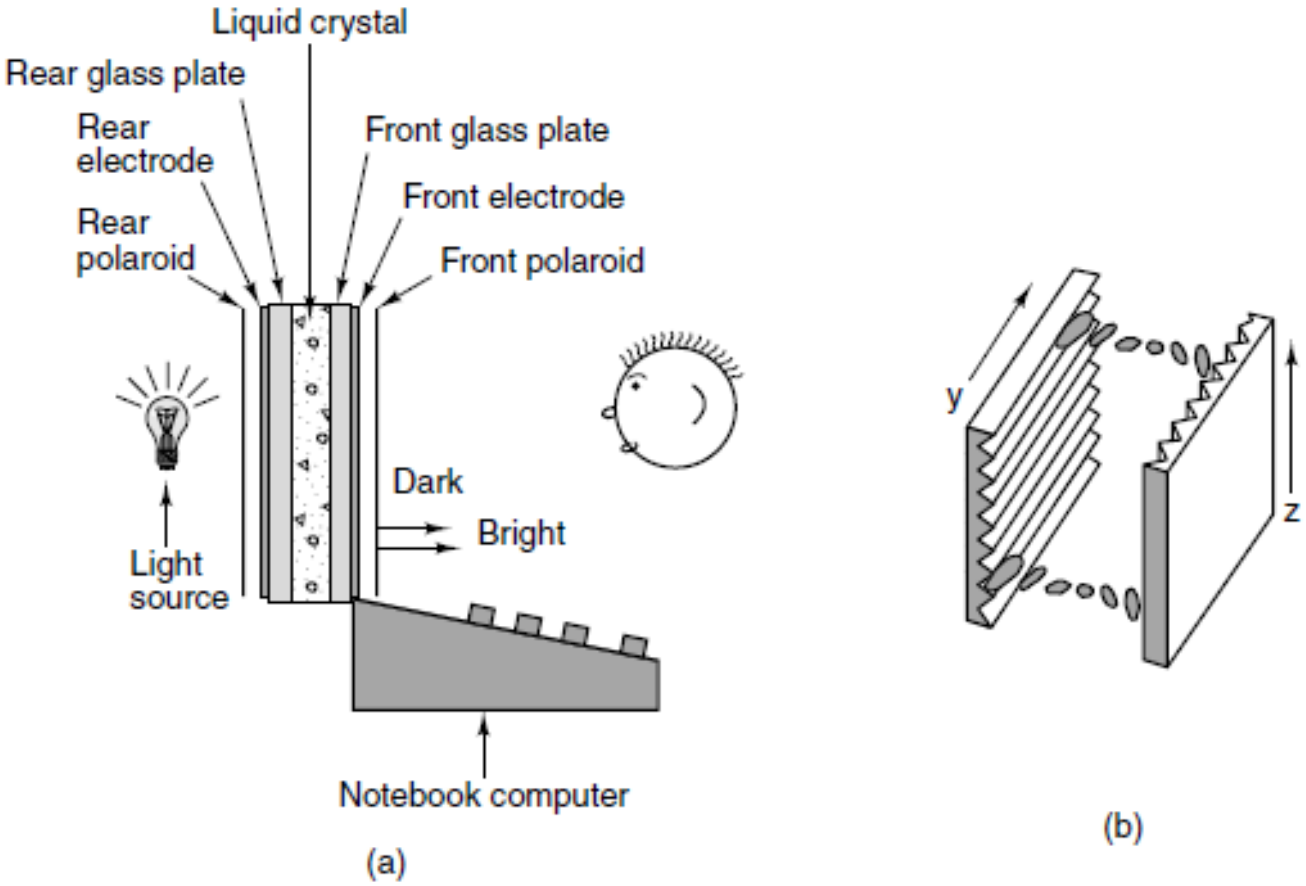
https://www.youtube.com/watch?v=pT1Ijg-v6mw

LCD

- Consists of two parallel glass plates, between which is a sealed volume containing a liquid crystal
- The rear plate contains tiny horizontal grooves and the front plate contains tiny vertical grooves
- The grooves on the rear and front plates are perpendicular to one another.

https://www.youtube.com/watch?v=k7xGQKpQAWw





The construction of an LCD screen. (b) The grooves on the rear and front plates are perpendicular to one another.

INPUT/OUTPUT

Mice

Design Purpose

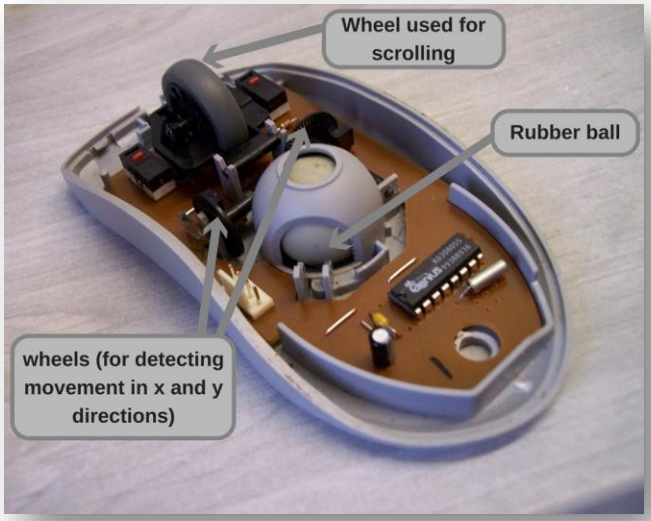
- Point to the menu items
- By people with less expertise in computers
- When it moves, a little pointer on the screen moves too, allow users to point at screen item

- **Three Kinds**

- Mechanical mice: wheel and ball
- Optical mice: no wheel and ball but LED (Light Emitting Diode)
- Optomechanical mice: newer mechanical mice with rolling ball

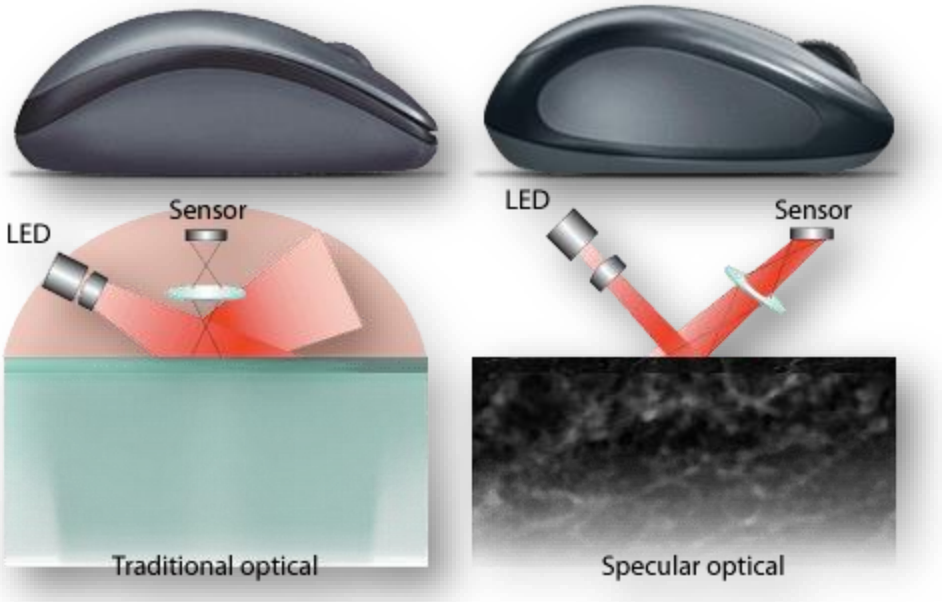
INPUT/OUTPUT

Mechanical mice: wheel and ball



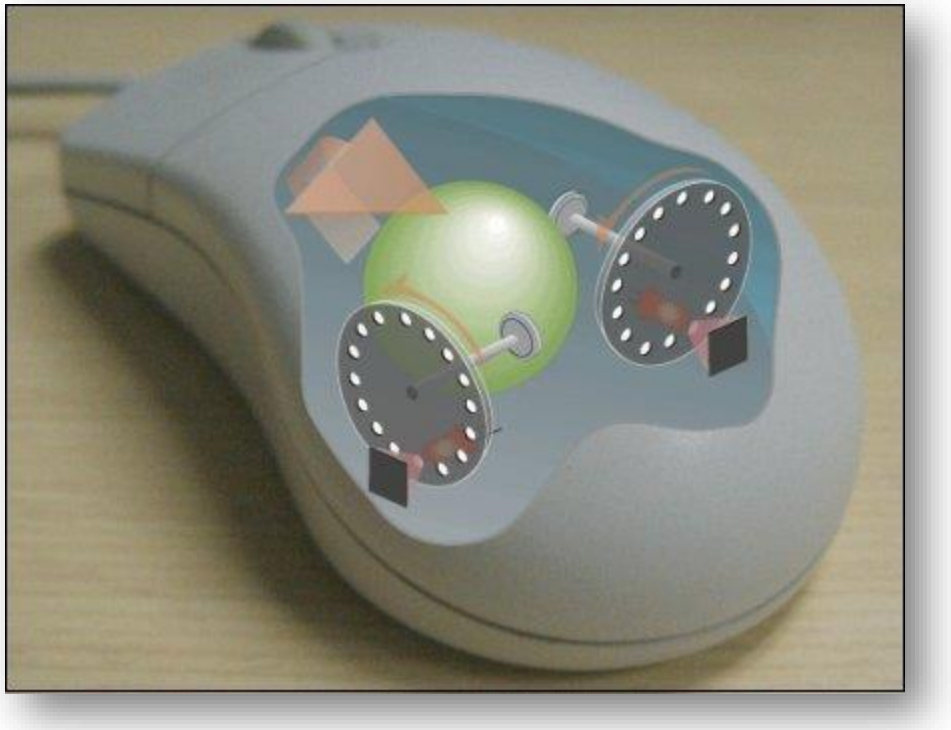
INPUT/OUTPUT

Optical mice



INPUT/OUTPUT

Optomechanical mice



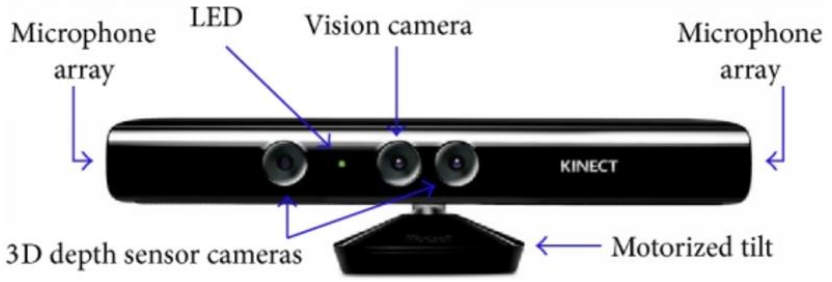
INPUT/OUTPUT

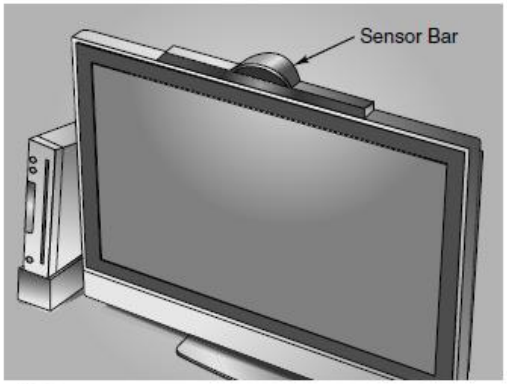
Game Controllers

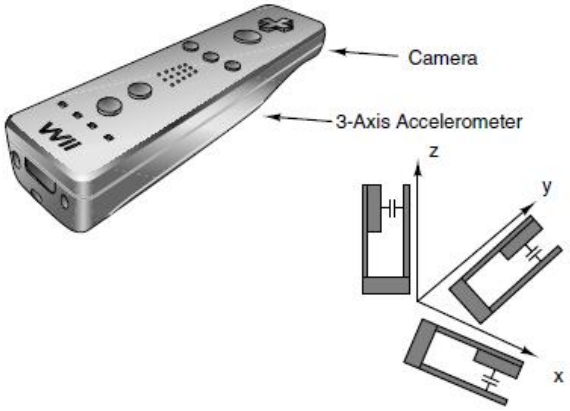
Wiimote Controller

Kinect Controller

Microsoft's motion sensor add-on for the Xbox 360 gaming console





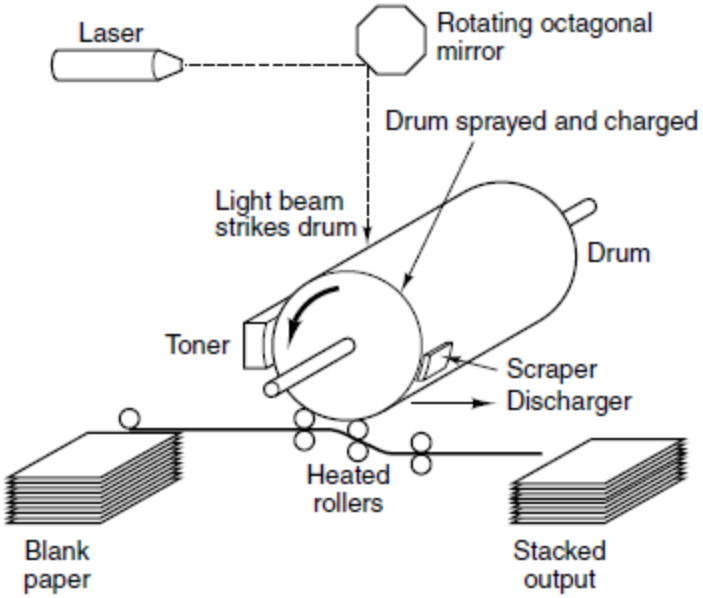


The Wiimote video game controller motion sensors.

INPUT/OUTPUT
Printers

Laser Printers

The heart is a rotating precision drum
– The printing scheme is an exceedingly complex combination of physics, chemistry, mechanical engineering and optical engineering



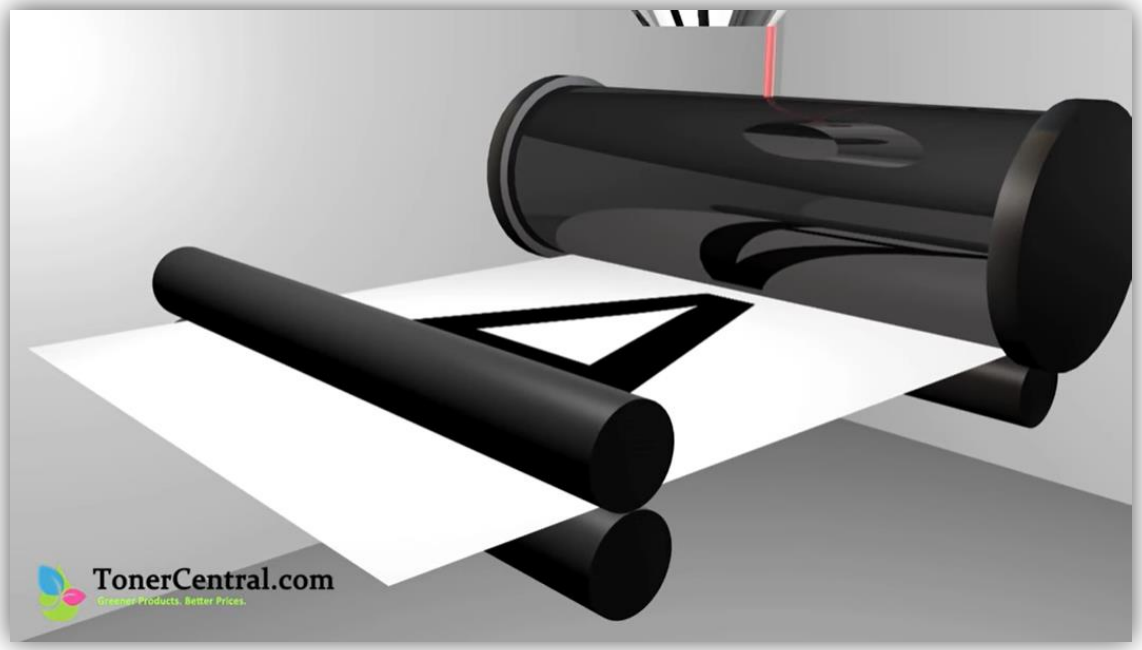
Operation of a laser printer.

INPUT/OUTPUT

Printers

How a laser printer works

https://www.youtube.com/watch?v=KtXes1sgUb4

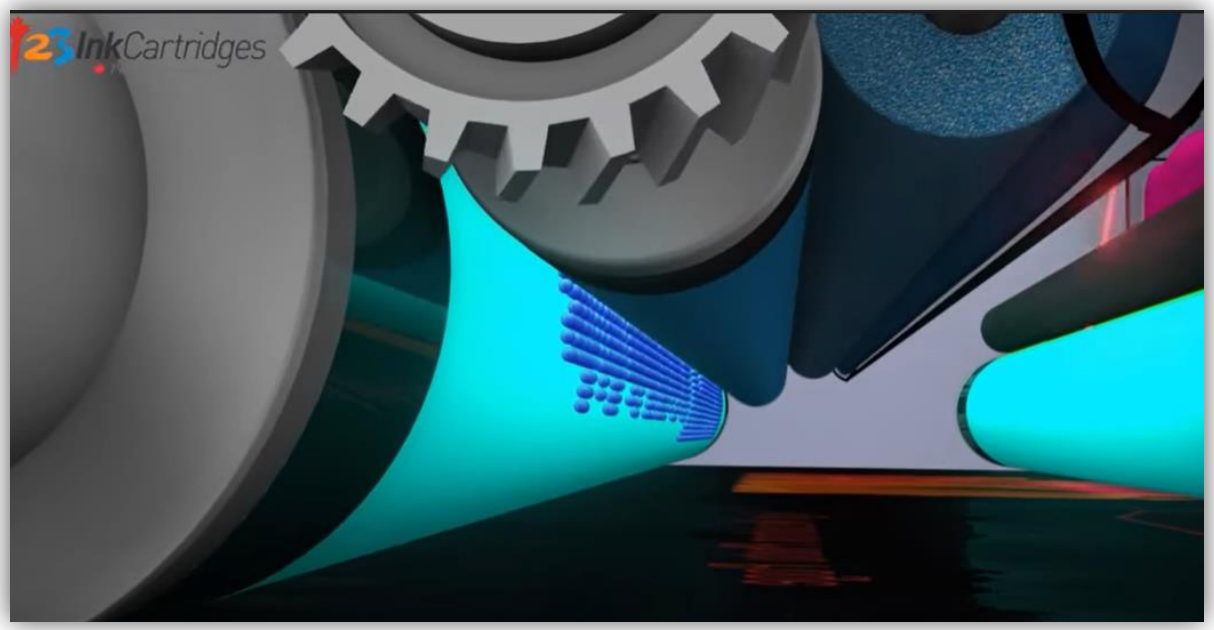


INPUT/OUTPUT

Printers

Color Printing

https://www.youtube.com/watch?v=C4nBCLOXwzE



INPUT/OUTPUT

Printers

Specialty Printers

- Thermal printer
- Inkjet printer
- Dye sublimation printer

https://www.youtube.com/watch?v=4HZVXU91Dtw

- Wax printer
- Solid ink printer

INPUT/OUTPUT

Digital Cameras

- **Charge-Coupled Devices (CCDs)**
  - A pixel is made up for 4 CCDs (one red, one blue, two green)
  - 6 millions CCDs is just for 1.5 millions pixels
- **Software Tasks in Camera**
  - Setting the focus
  - Determine exposure
  - Performing the white balance
  - Possibly localizing human faces

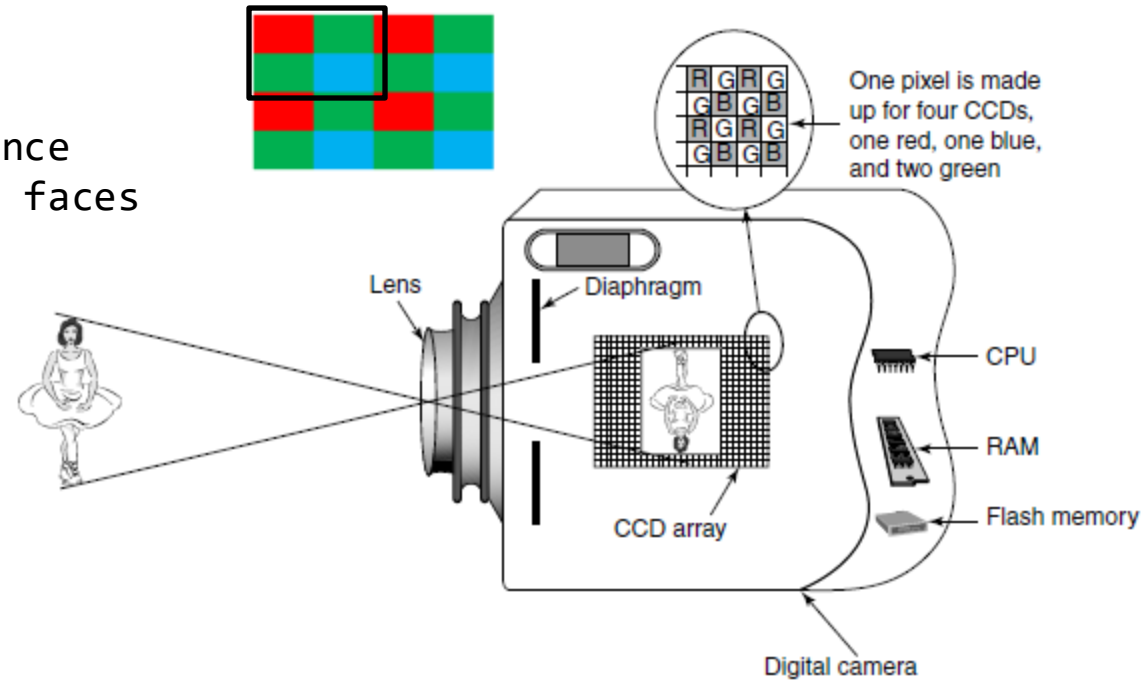


Figure 2-43. A digital camera.

INPUT/OUTPUT

Telecommunications Equipment

- Network
- Most computers are connected to a computer network, often internet

- Equipment's

- Modems
- Digital Subscriber Lines (DSL)
- Internet over Cable

INPUT/OUTPUT

Character Codes

Character set

- 26 uppercase letters (A – Z)
- 26 lowercase letters (a – z)
- Digits 0 through 9
- Special symbols, such as space, period, minus sign, comma, and carriage return

ASCII (American Standard Code for Information Interchange)

7 bits is used allowing for 128(2^7) characters

INPUT/OUTPUT

Character Codes

Hex	Name	Meaning	Hex	Name	Meaning
0	NUL	Null	10	DLE	Data Link Escape
1	SOH	Start Of Heading	11	DC1	Device Control 1
2	STX	Start Of TeXt	12	DC2	Device Control 2
3	ETX	End Of TeXt	13	DC3	Device Control 3
4	EOT	End Of Transmission	14	DC4	Device Control 4
5	ENQ	Enquiry	15	NAK	Negative AcKnowledgement
6	ACK	ACKnowledgement	16	SYN	SYNchronous idle
7	BEL	BELl	17	ETB	End of Transmission Block
8	BS	BackSpace	18	CAN	
9	HT	Horizontal Tab	19	EM	
A	LF	Line Feed	1A	SUB	
B	VT	Vertical Tab	1B	ESC	
C	FF	Form Feed	1C	FS	
D	CR	Carriage Return	1D	GS	
E	SO	Shift Out	1E	RS	
F	SI	Shift In	1F	US	

Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char
20	(Space)	30	0	40	@	50	P	60	‘	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	"	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	’	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

INPUT/OUTPUT

Character Codes

Unicode

16 bits is used allowing for 65536(2^{16}) code points

Universal Character Set Transformation Format (UTF-8)

Can code about two billion characters and variable length, from 1 to 4 bytes

Bits	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	0ddddddd					
11	110ddddd	10dddddd				
16	1110dddd	10dddddd	10dddddd			
21	11110ddd	10dddddd	10dddddd	10dddddd		
26	111110dd	10dddddd	10dddddd	10dddddd	10dddddd	
31	1111110x	10dddddd	10dddddd	10dddddd	10dddddd	10dddddd

The UTF-8 encoding scheme.

± ASCII 177 - 10110001

UTF-8 11000010 10110001 (0xC2B1)
10110001

中 20013

UTF-8 11100100 10111000 10101101 (0xE4B8AD)
100111000101101

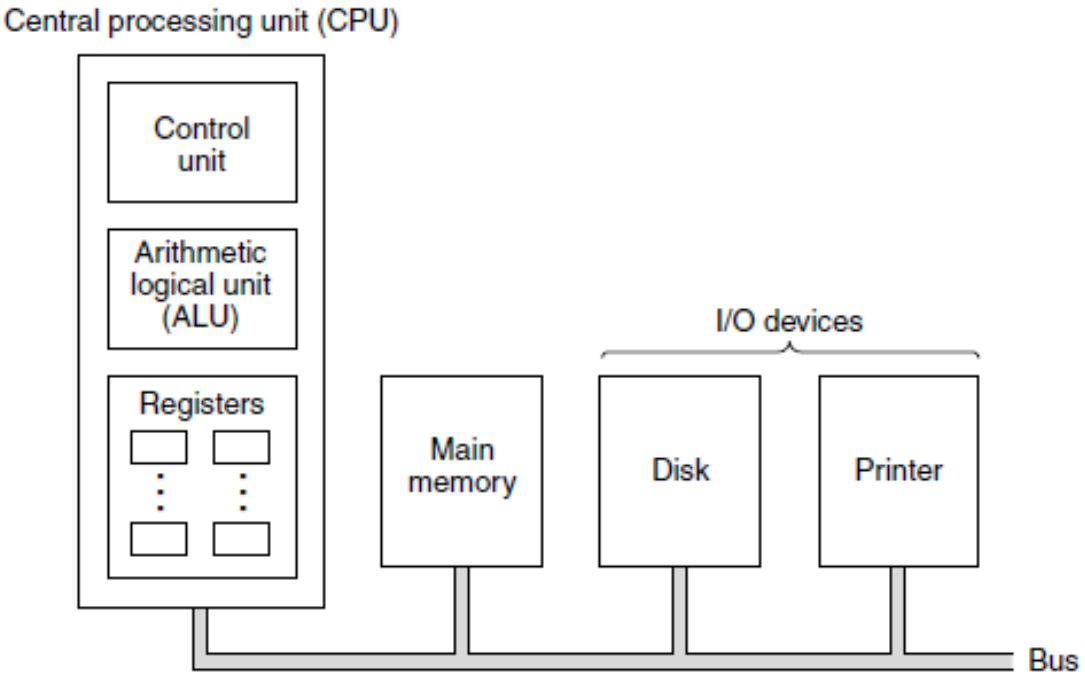
Bits	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	0ddddddd					
11	110ddddd	10dddddd				
16	1110dddd	10dddddd	10dddddd			
21	11110ddd	10dddddd	10dddddd	10dddddd		
26	111110dd	10dddddd	10dddddd	10dddddd	10dddddd	
31	1111110x	10dddddd	10dddddd	10dddddd	10dddddd	10dddddd

The UTF-8 encoding scheme.

Processors
Primary Memory
Secondary Memory
Input/Output
Microarchitecture level

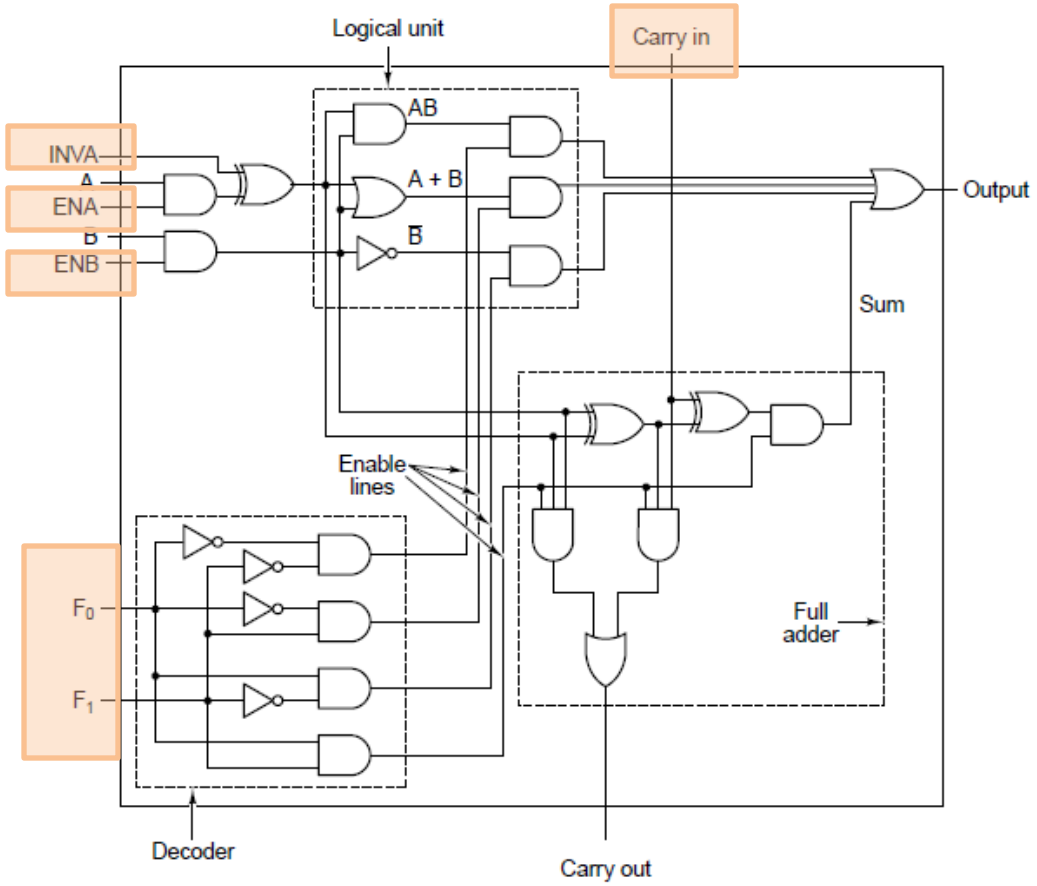
Microarchitecture level

- **CPU**
  - ALU (arithmetic logic unit)
  - Registers
  - **Control unit**



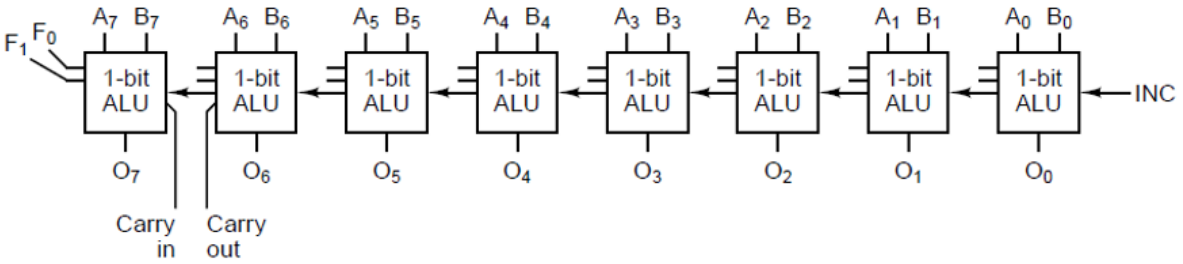
ALU

1 bit ALU



Six control signals

8-bit ALU



ALU

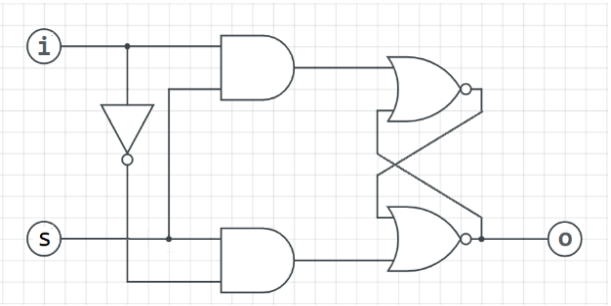
Six Control Signals

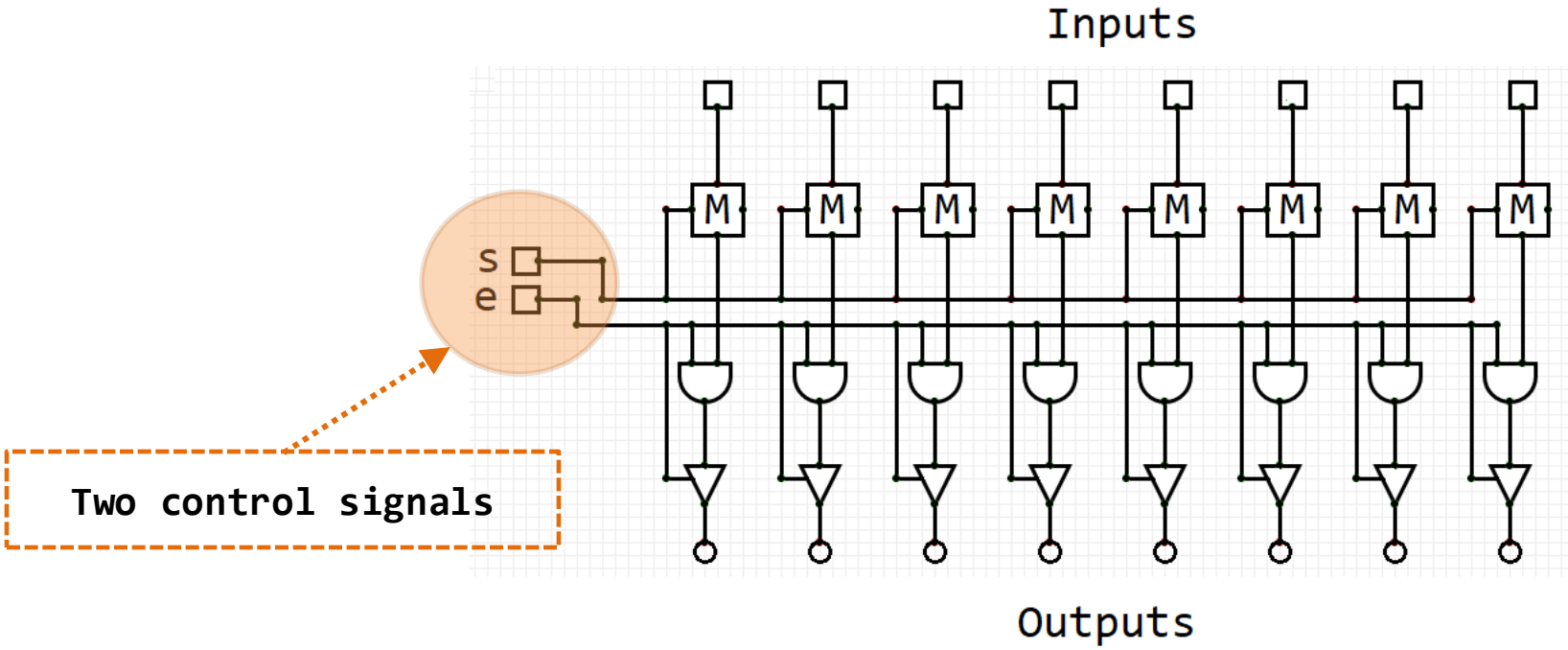
F_0	F_1	ENA	ENB	INVA	INC	Function	
0	1	1	0	0	0	A	
0	1	0	1	0	0	B	
0	1	1	0	1	0	$\bar{A}$	NOT
1	0	1	1	0	0	$\bar{B}$	
1	1	1	1	0	0	A + B	ADD
1	1	1	1	0	1	A + B + 1	
1	1	1	0	0	1	A + 1	INC
1	1	0	1	0	1	B + 1	
1	1	1	1	1	1	B – A	SUB
1	1	0	1	1	0	B – 1	DEC
1	1	1	0	1	1	–A	NEG
0	0	1	1	0	0	A AND B	AND
0	1	1	1	0	0	A OR B	OR
0	1	0	0	0	0	0	
1	1	0	0	0	1	1	
1	1	0	0	1	0	–1	

Operations

Registers

1 bit memory





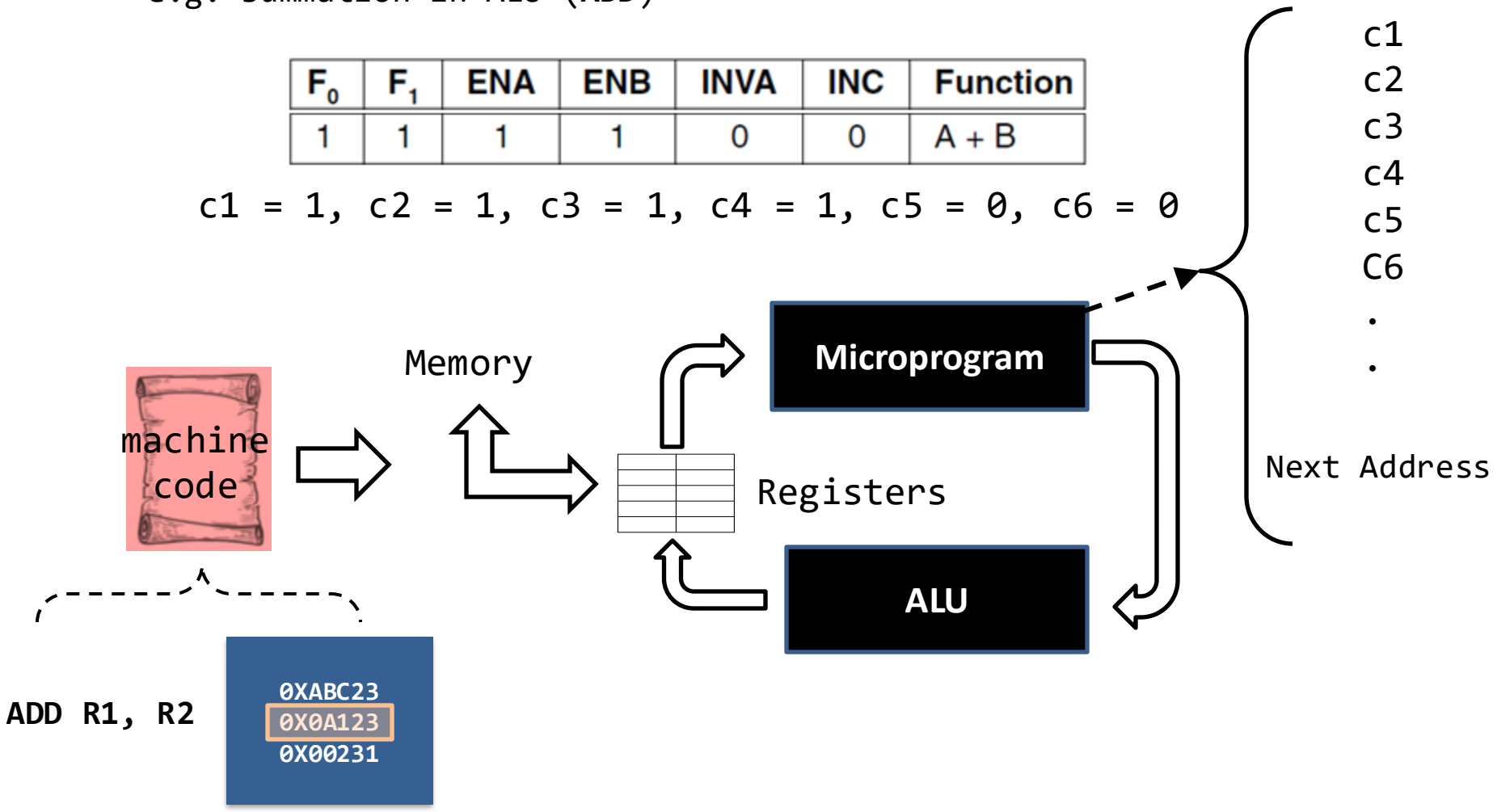
8-bit memory (Register)

- Control signals in general are shown by c1, c2, c3, c4, ...
- They are generated by control unit
- A high level programming language is required to create right instructions for CPU (Assembly, C, ...)

e.g. summation in ALU (**ADD**)

F_0	F_1	ENA	ENB	INVA	INC	Function
1	1	1	1	0	0	A + B

c1 = 1, c2 = 1, c3 = 1, c4 = 1, c5 = 0, c6 = 0



Microinstruction : an array of control signals

Microprogram : a set of microinstructions

c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20		
P0	S1	E1	S2	E2	S3	E3	A	B	ENB_B	ENB_A	INVA	Cin	F0	F1	SACC	ACC	A0	A1	m0	BIN	HEX
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	10100001000000000000	0xA1000
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10100000000000000000	0xA0000
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	10001000100000000000	0x88800
1	0	0	0	1	0	0	0											0	0	10001000000000000000	0x88000
1	0	0	0	0	1	0	0											0	0	10000100011001111000	0x84678
1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	1	0	0	0	10000000011001101000	0x80668

Register level operations

When an instruction such as ADD, LOAD, SUB, . . . is read a sequence of microinstructions is initiated and then executed

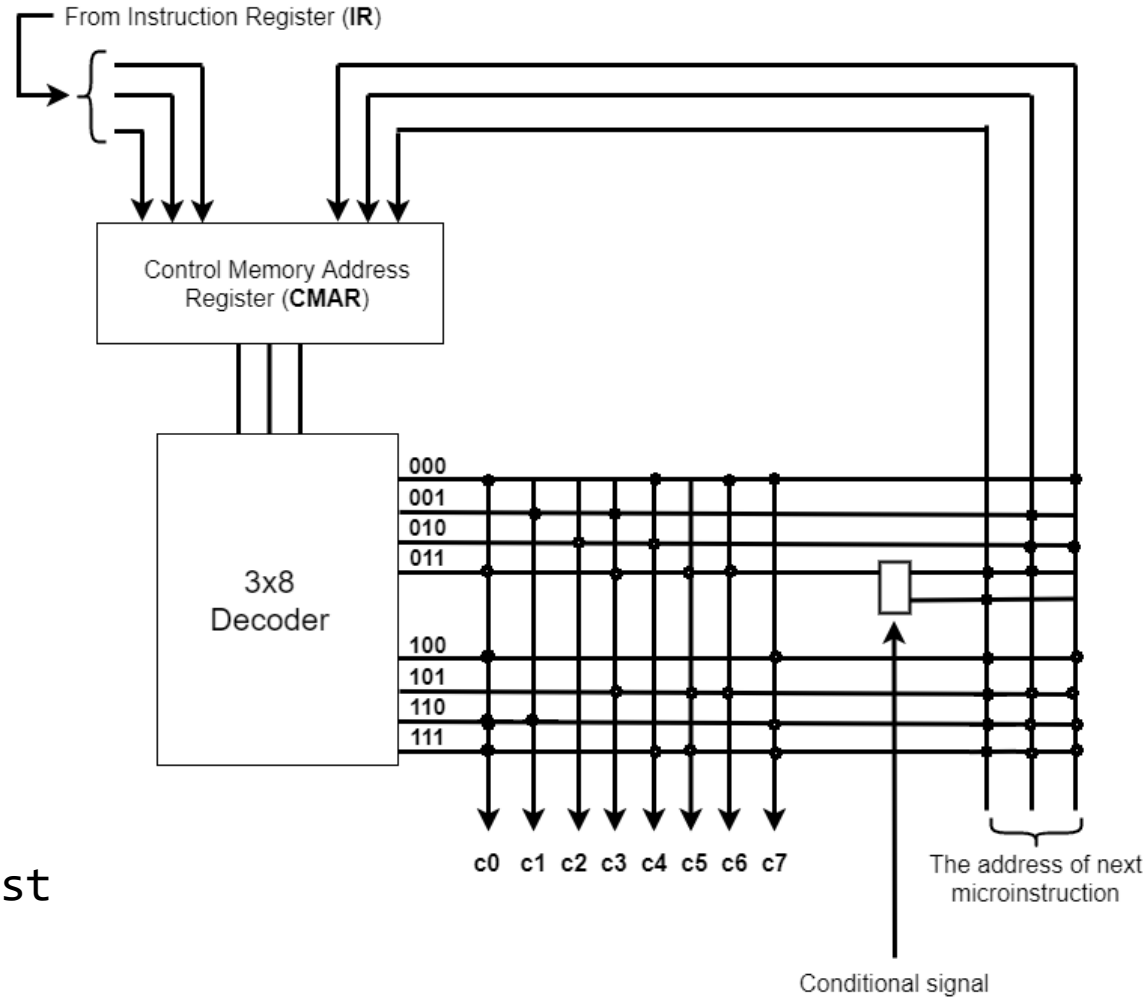
ADD

c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
P0	S1	E1	S2	E2	S3	E3	A	B	ENB_B	ENB_A	INVA	Cin	F0	F1	SACC	ACC	A0	A1	m0
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	1	1	0	0	1	1	1	1	0	0	0
1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	1	0	0	0

Microprogrammed Control

Maurice Wilkes
British computer scientist

1913 - 2010

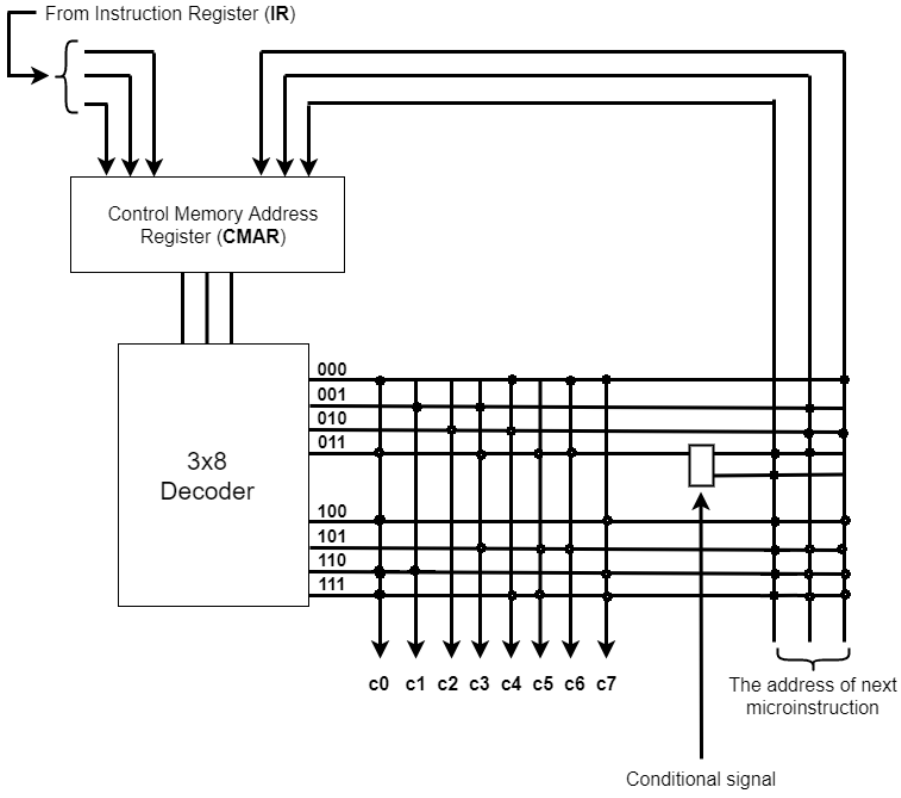


Wilkes Control Unit

Microprogrammed Control

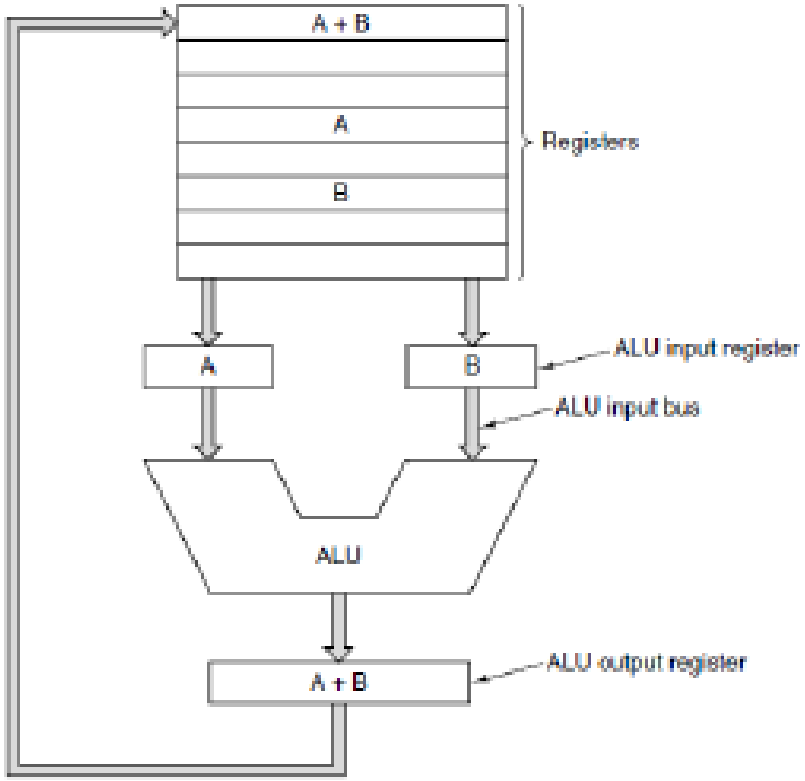
https://www.youtube.com/watch?v=WsGAo-_rHSo

Activated line	Generated control signals	Next address
0 0 0	C0 C4 C6 C7	0 0 1
0 0 1	C1 C3	0 1 0
0 1 0	C2 C4	0 1 1
0 1 1	C0 C3 C5 C6	
0 1 1	C0 C3 C5 C6	If True then 110
1 1 0	C0 C1 C7	1 1 1
1 1 1	C0 C4 C5 C7	Next Instruction in IR
0 1 1	C0 C3 C5 C6	If False then 100
1 0 0	C0 C7	1 0 1
1 0 1	C3 C5 C6	1 1 1
1 1 1	C0 C4 C5 C7	Next Instruction in IR



Data Path Cycle

The process of running two operands through the ALU and storing results



Mic-1

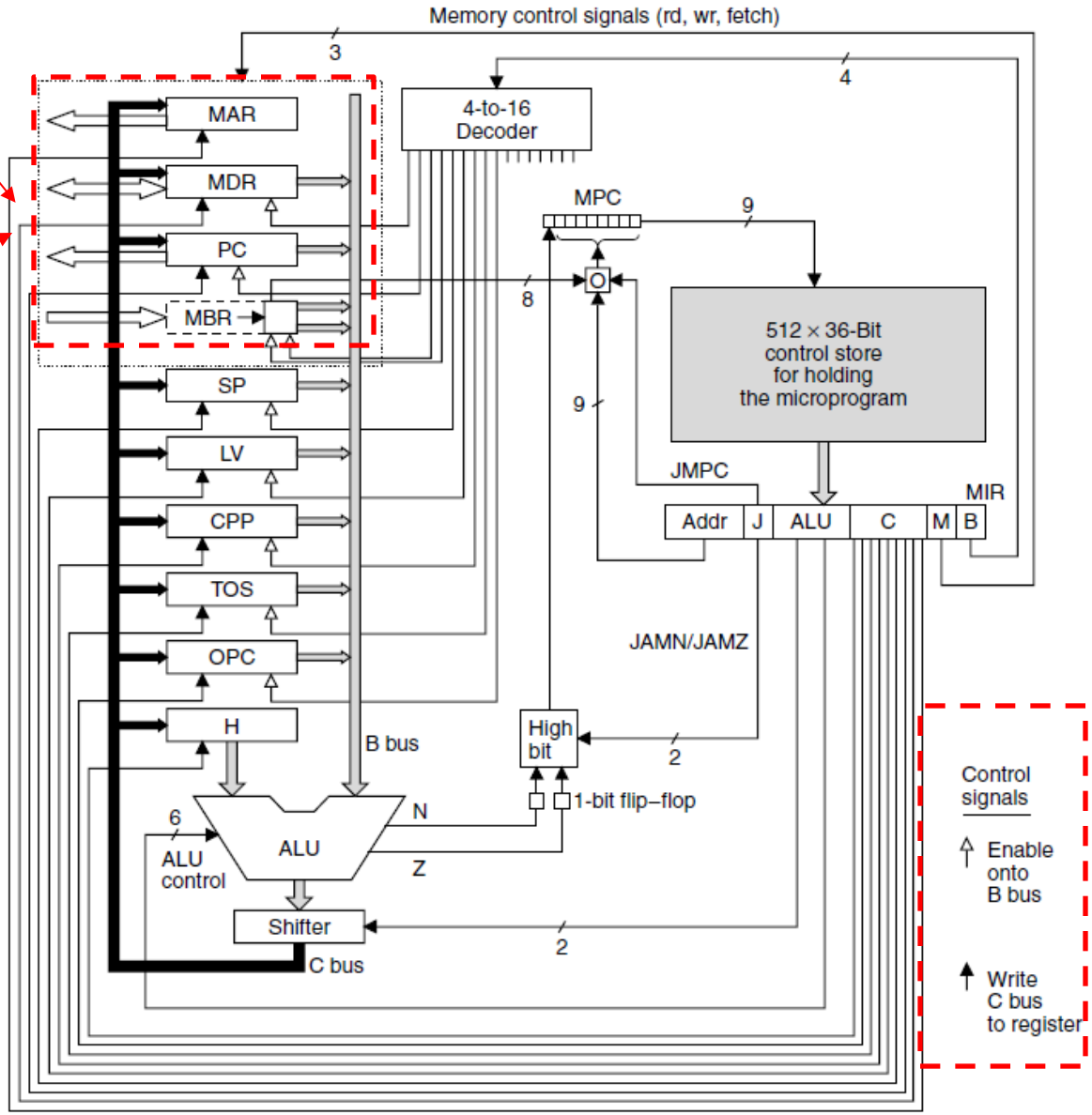
- Stack based
- ISA is a subset of the Java Virtual Machine (IJVM)
- IJVM instructions are short and sweet
- IJVM is small, it is a good starting point for describing the control and sequencing of instructions

IJVM = Integer Java Virtual Machine
ISA = Instruction Set Architecture

Microprogrammed Control

Memory control registers

To and from Main memory



MPC = MicroProgram Counter

MIR = MicroInstruction Register

Buses

A Bus - Only contents of register **H** can go on this bus and it serves as left input of ALU

B Bus - serves as right input to ALU; most registers can put their contents on this bus

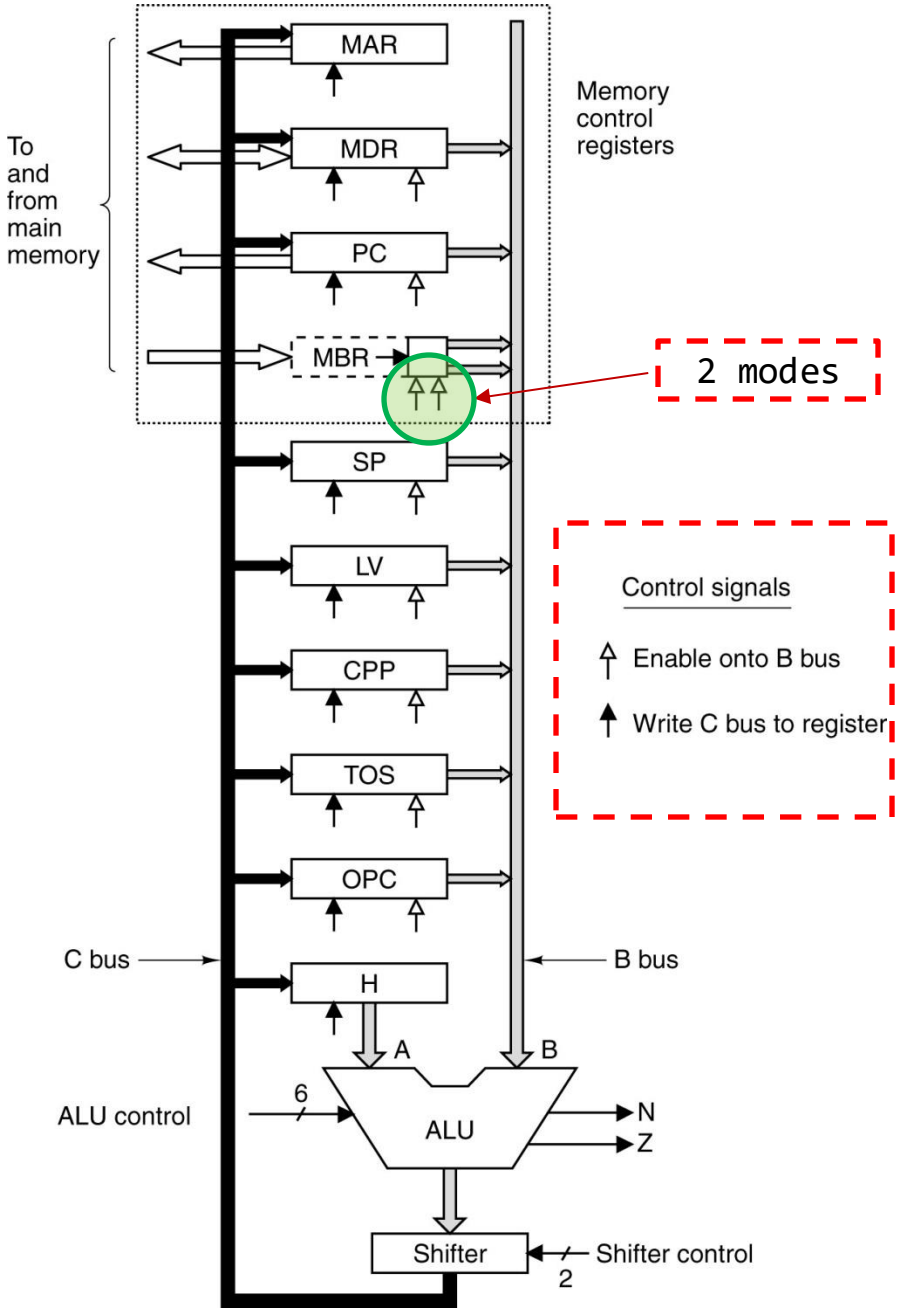
C Bus - carries ALU output back into registers

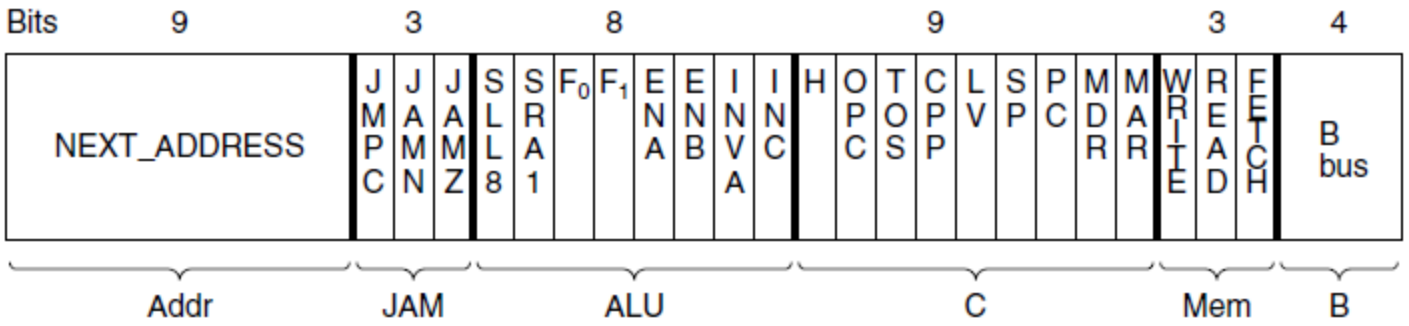
Registers

32 bit registers

MAR - Memory Address register
MDR - Memory Data Register
PC - Program Counter
MBR - Memory Buffer Register
SP - Stack Pointer
LV - Local Variable pointer
CPP - Constant Pool Pointer
TOS - Top of Stack register
OPC - OpCode register
H - Holding Register is the left ALU input.

ALU has 6 control lines and two outputs
Shifter has 2 control lines
Not shown are 3 memory control lines





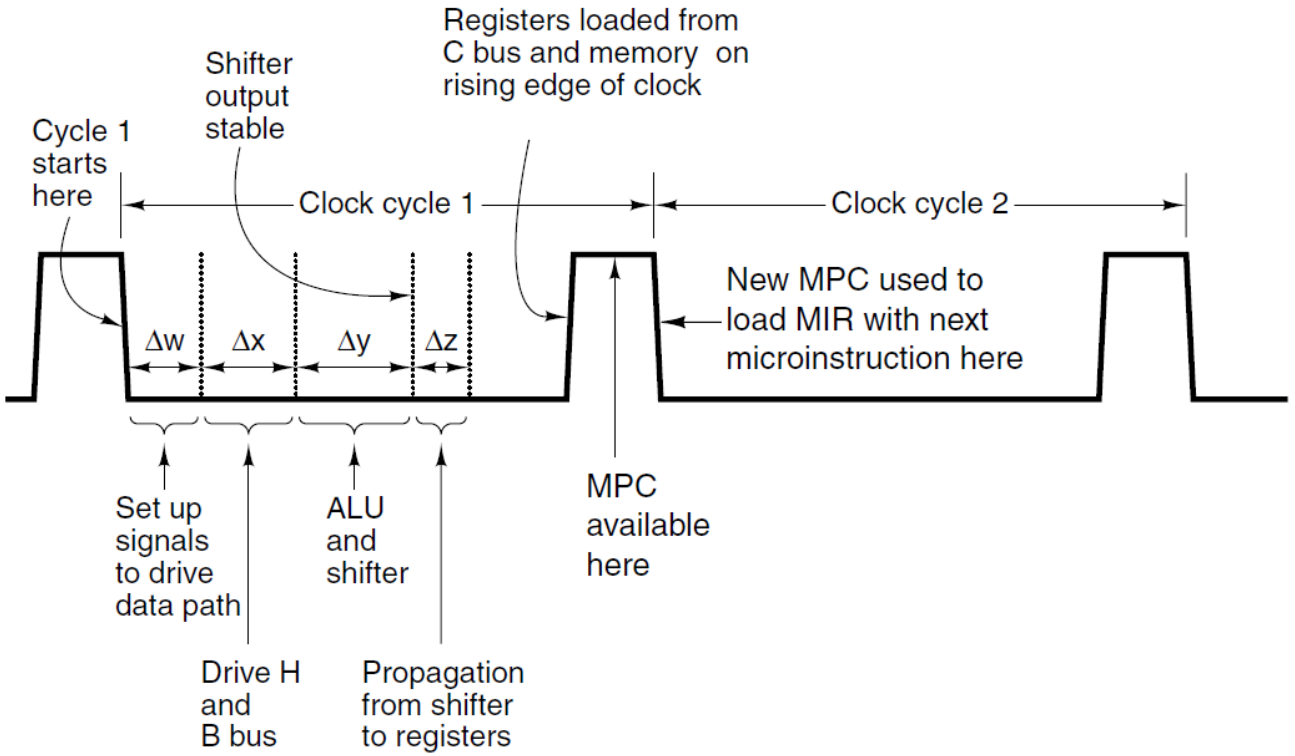
B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

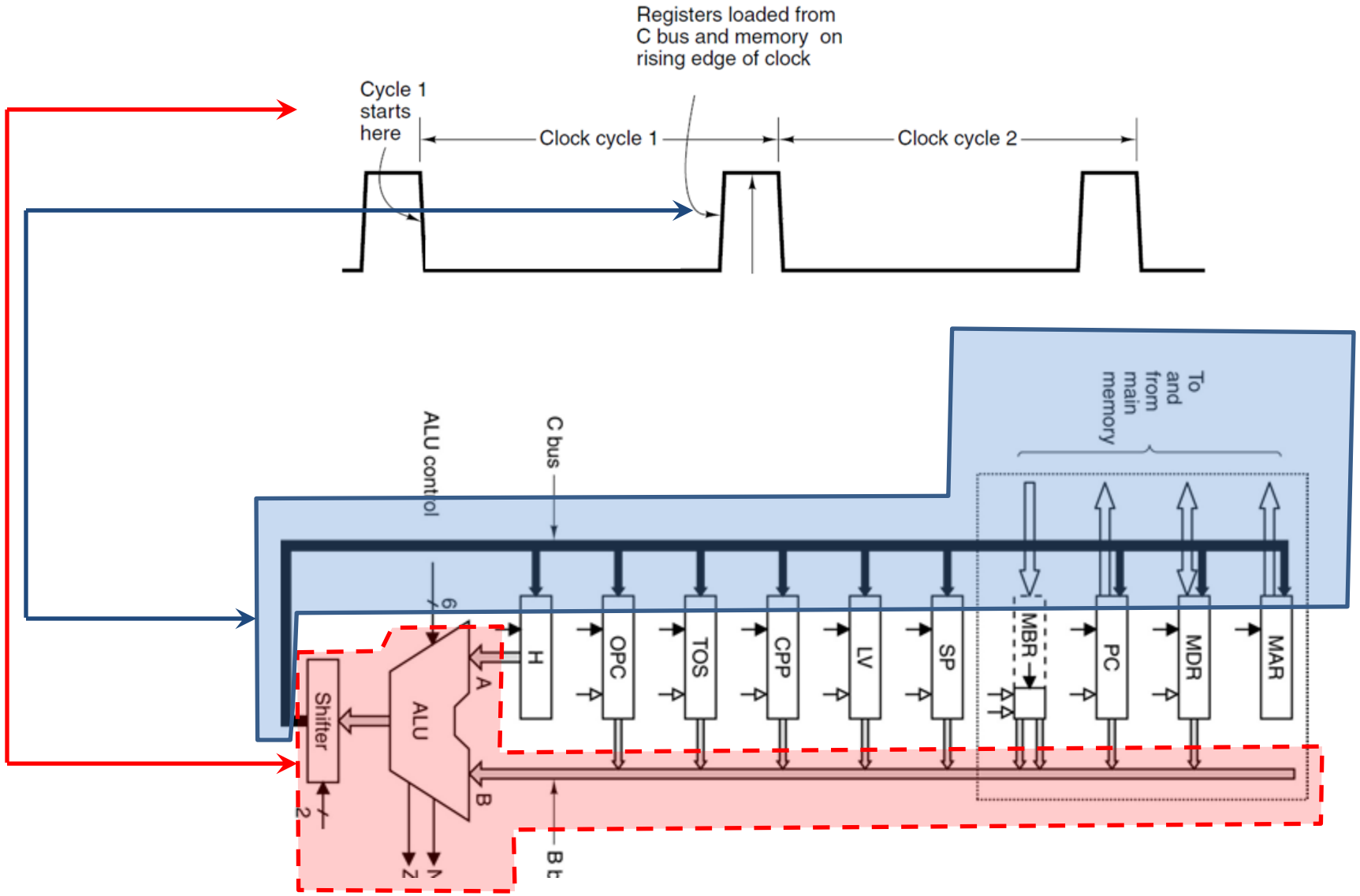
The microinstruction format for the Mic-1

Data Path Timing

1. The control signals are set up (Δw).
2. The registers are loaded onto the B bus (Δx).
3. The ALU and shifter operate (Δy).
4. The results propagate along the C bus back to the registers (Δz).



Data Path Timing



Instruction sets

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO *offset*	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal
0x84	IINC *varnum const*	Add a constant to a local variable
0x15	ILOAD *varnum*	Push local variable onto stack
0xB6	INVOKEVIRTUAL *disp*	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE *varnum*	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

- Local Variable indices:
  - i 1
  - j 2
  - k 3

```
i = j + k;                  1        ILOAD j          // i = j + k        0x15 0x02
if (i == 3)                 2        ILOAD k                           0x15 0x03
    k = 0;                  3        IADD                              0x60
else                        4        ISTORE i                          0x36 0x01
   j = j – 1;               5        ILOAD i          // if (i == 3)    0x15 0x01
                            6        BIPUSH 3                          0x10 0x03
                            7        IF_ICMPEQ L1                      0x9F 0x00 0x0D
                            8        ILOAD j          // j = j – 1      0x15 0x02
                            9        BIPUSH 1                          0x10 0x01
                           10        ISUB                              0x64
                           11        ISTORE j                          0x36 0x02
                           12        GOTO L2                           0xA7 0x00 0x07
                           13   L1:  BIPUSH 0         // k = 0          0x10 0x00
                           14        ISTORE k                          0x36 0x03
                           15   L2:
```

(a) (b) (c)

Figure 4-14. (a) A Java fragment. (b) The corresponding Java assembly language. (c) The IJVM program in hexadecimal.

Microprogram

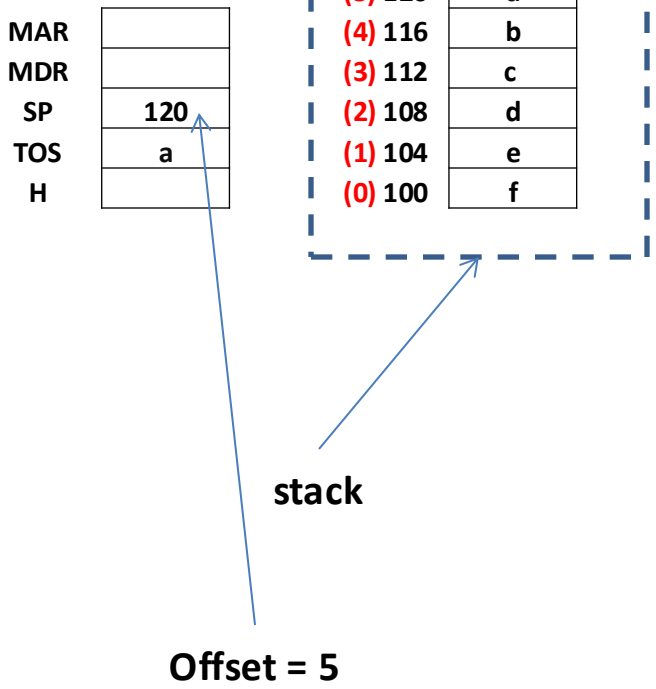
Only 112 microinstructions total

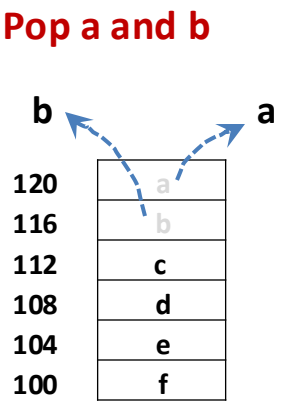
Label	Operations	Comments
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch
nop1	goto Main1	Do nothing
iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack
isub1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
isub2	H = TOS	H = top of stack
isub3	MDR = TOS = MDR – H; wr; goto Main1	Do subtraction; write to top of stack
iand1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iand2	H = TOS	H = top of stack
iand3	MDR = TOS = MDR AND H; wr; goto Main1	Do AND; write to new top of stack
ior1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
ior2	H = TOS	H = top of stack
ior3	MDR = TOS = MDR OR H; wr; goto Main1	Do OR; write to new top of stack
dup1	MAR = SP = SP + 1	Increment SP and copy to MAR
dup2	MDR = TOS; wr; goto Main1	Write new stack word
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
pop2		Wait for new TOS to be read from memory
pop3	TOS = MDR; goto Main1	Copy new word to TOS

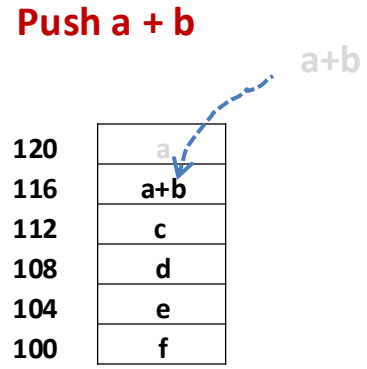
IADD:

1. TOS is already present, but the next-to-top word of the stack must be fetched from memory.
2. TOS must be added to the next-to-top word fetched from memory.
3. The result, which is to be pushed on the stack, must be stored back into memory, as well as stored in the TOS register.

0x60	IADD	Pop two words from stack; push their sum

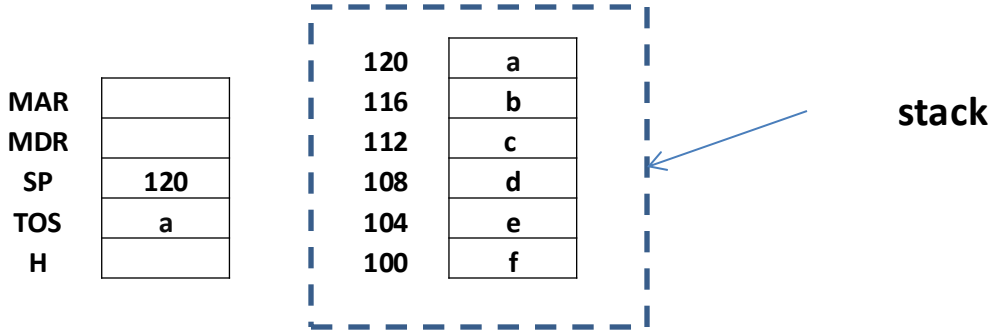






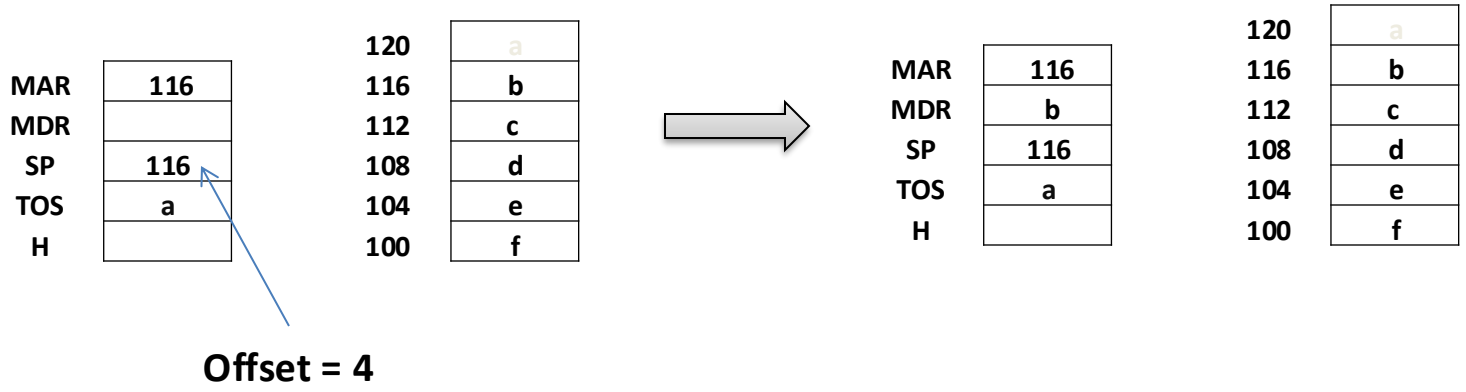
iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack

iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack

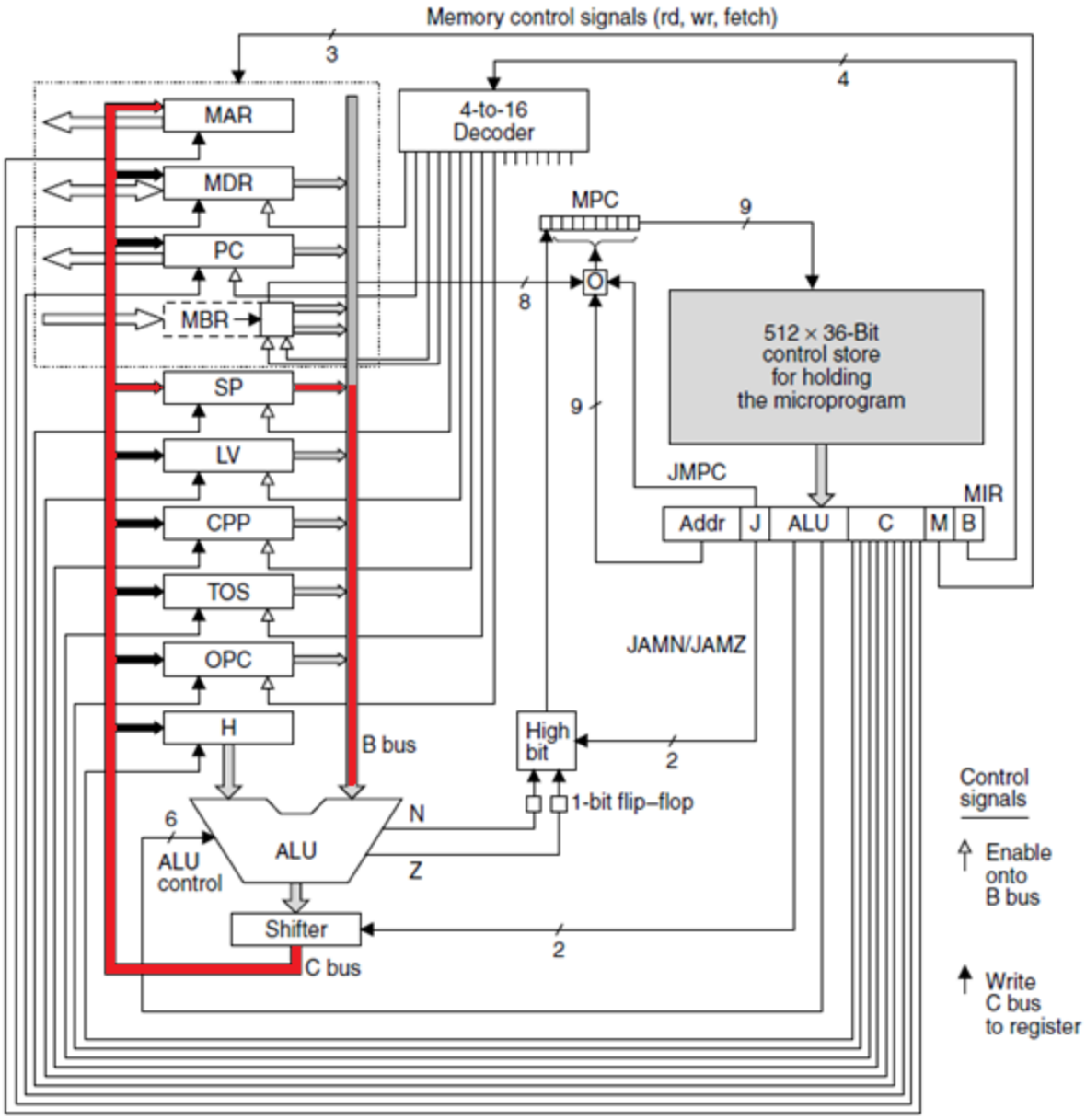


iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack

MAR = SP=SP-1 **rd**

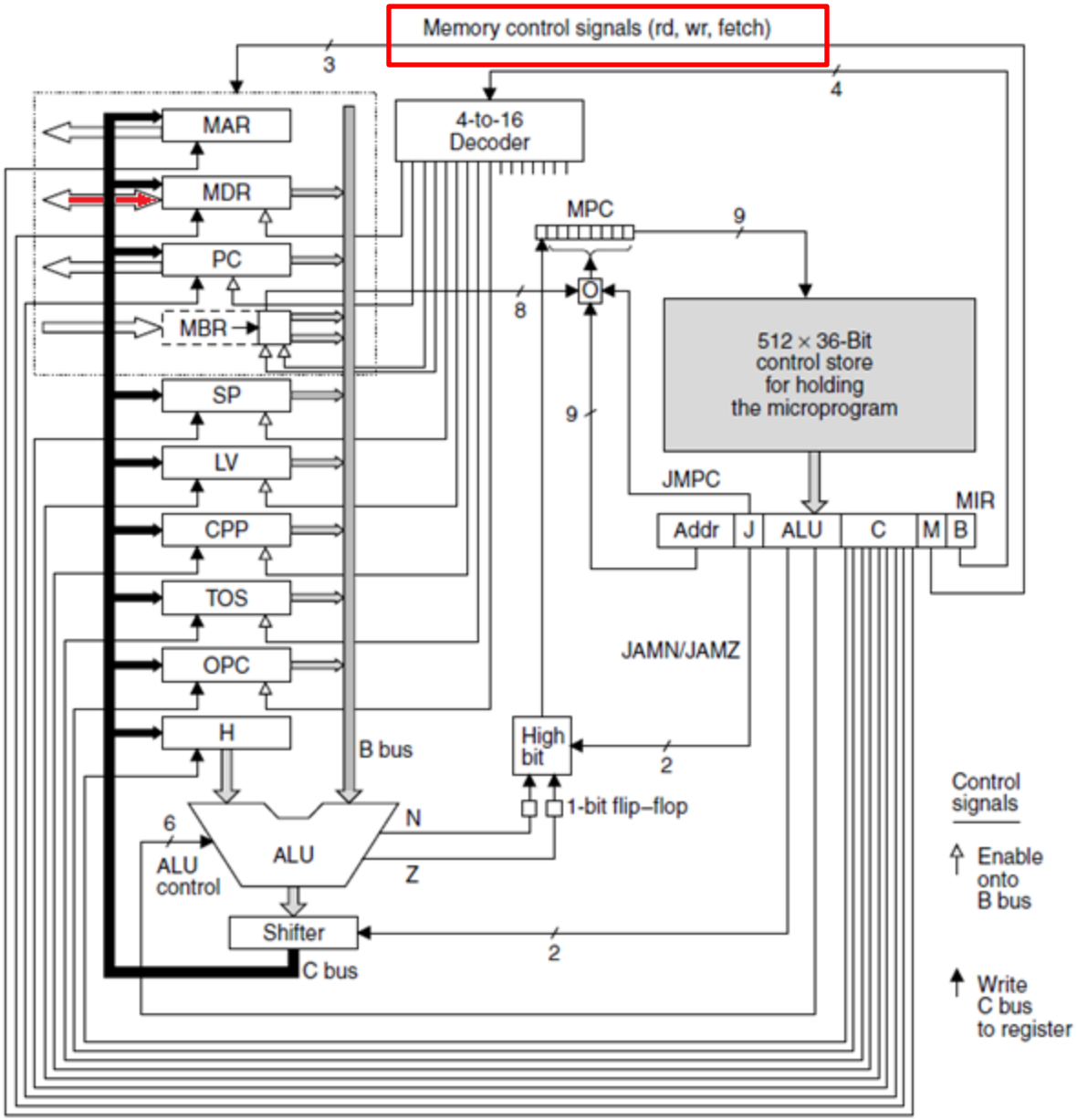


MAR = SP=SP-1



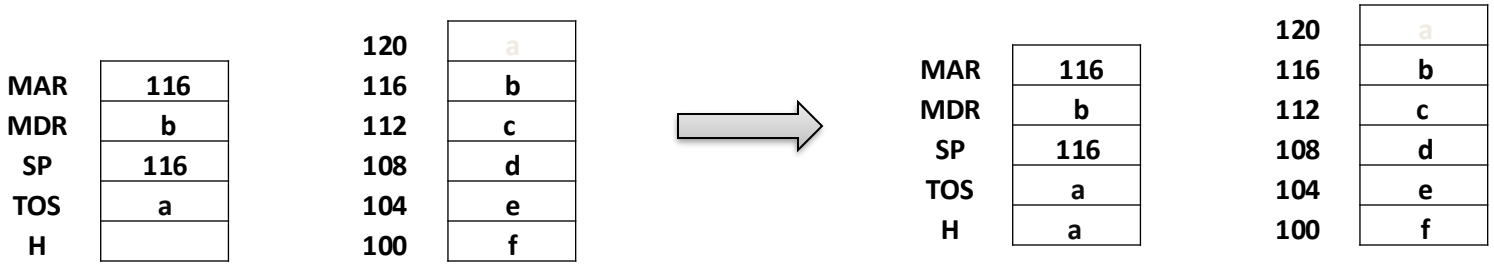
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

rd

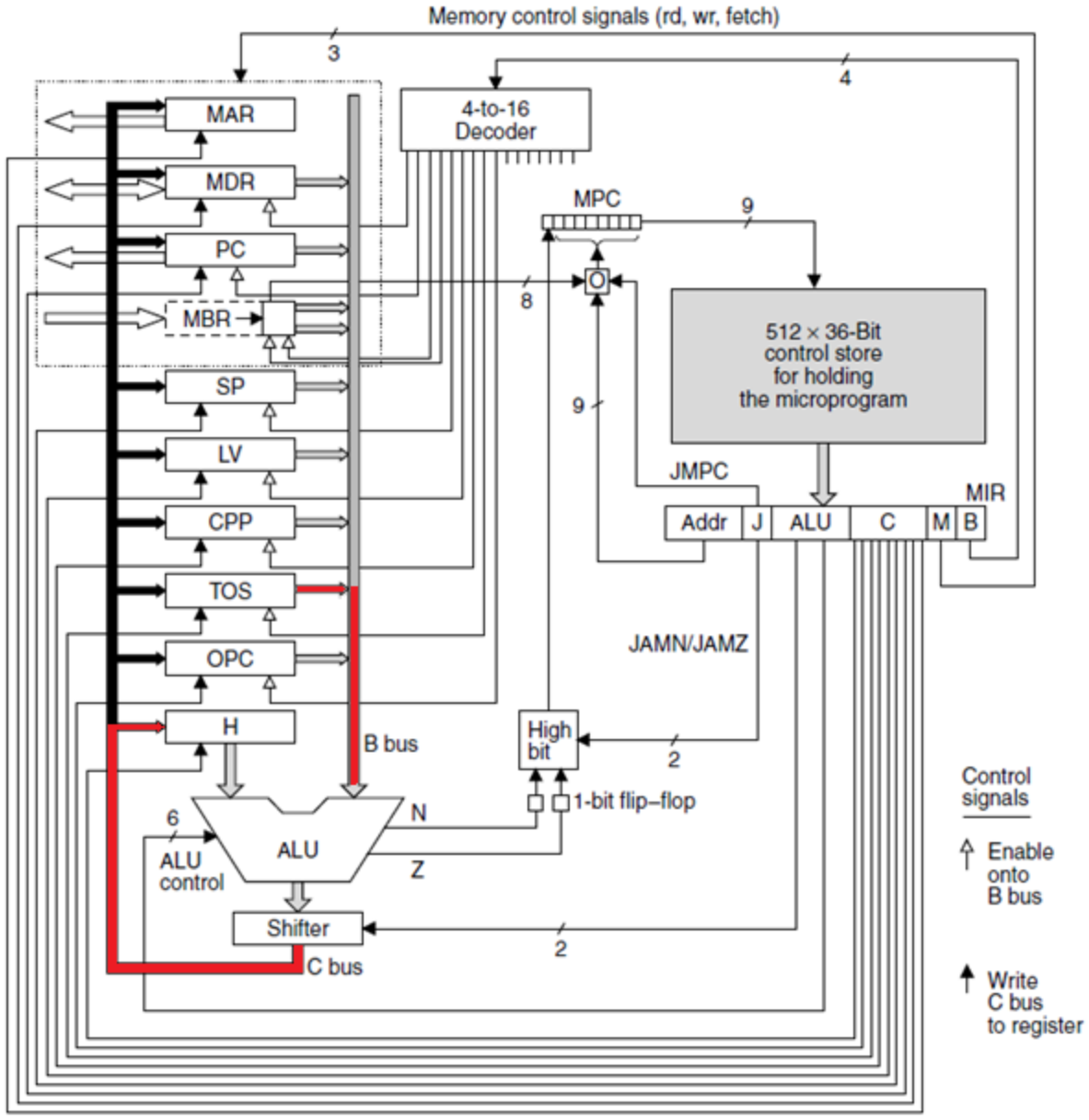

Memory control signals (rd, wr, fetch)
3
4
4-to-16 Decoder
MAR
MDR
PC
MBR
SP
LV
CPP
TOS
OPC
H
MPC
9
8
9
512 × 36-Bit control store for holding the microprogram
JMPC
MIR
Addr
J
ALU
C
M
B
JAMN/JAMZ
High bit
2
1-bit flip–flop
B bus
6
ALU control
ALU
N
Z
Shifter
2
C bus
Control signals
Enable onto B bus
Write C bus to register

iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack

H=TOS



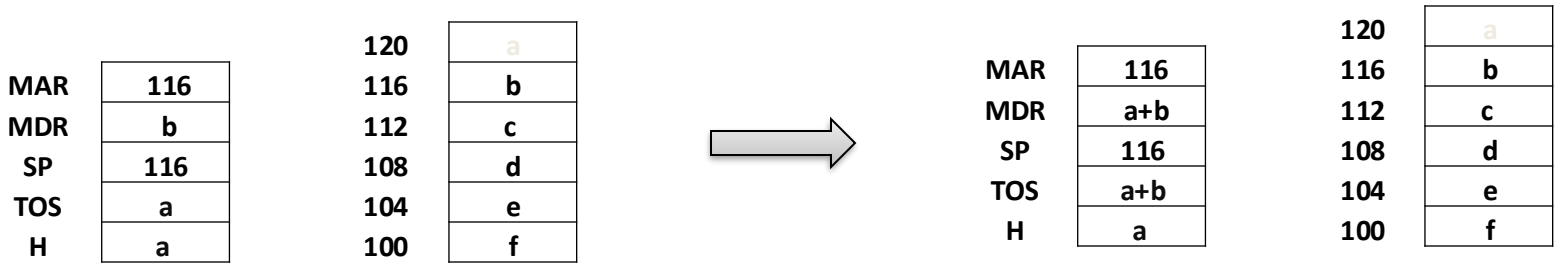
H=TOS



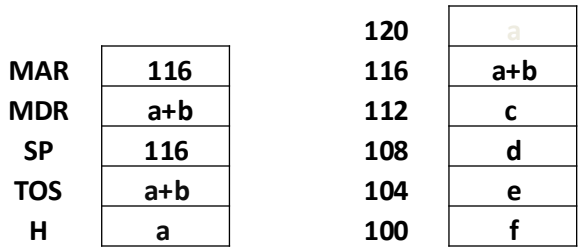
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack

MDR=TOS=MDR+H



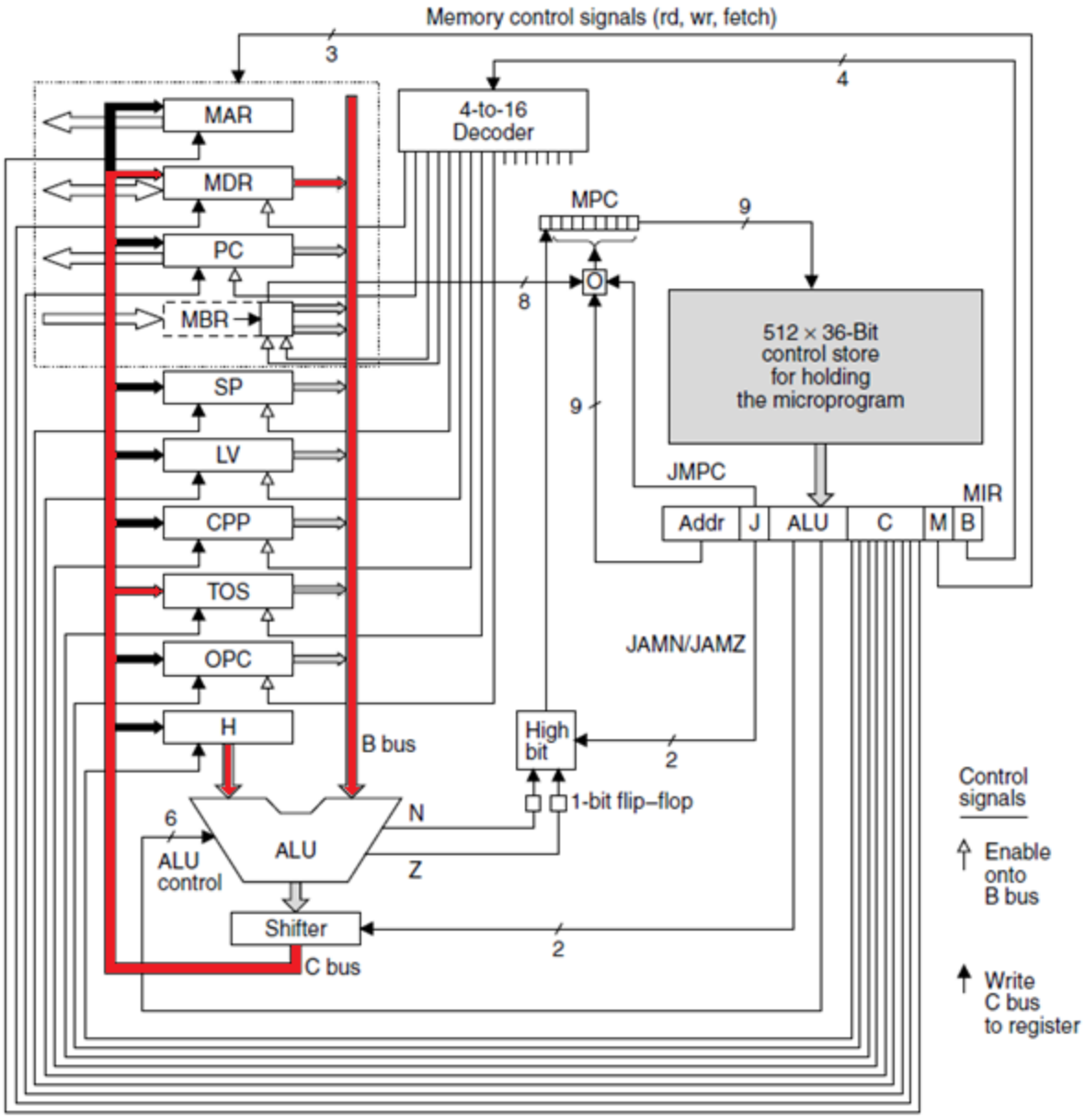
wr



goto Main1

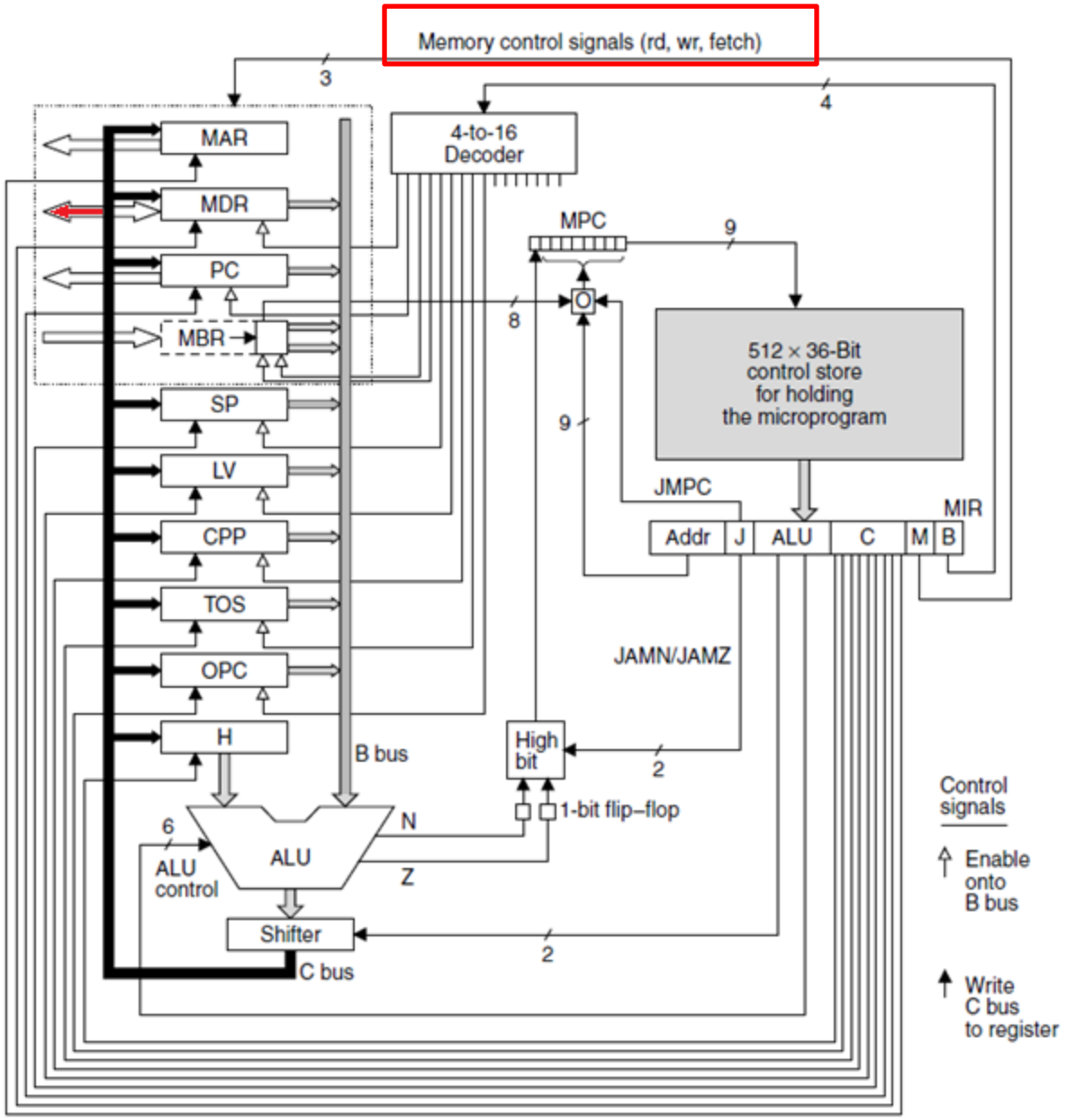
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch

MDR=TOS=MDR+H



F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

wr


Memory control signals (rd, wr, fetch)
3
4
4-to-16 Decoder
MAR
MDR
PC
MBR
SP
LV
CPP
TOS
OPC
H
MPC
9
8
9
512 × 36-Bit control store for holding the microprogram
JMPC
MIR
Addr
J
ALU
C
M
B
JAMN/JAMZ
High bit
2
B bus
1-bit flip–flop
N
Z
6
ALU control
ALU
Shifter
2
C bus
Control signals
Enable onto B bus
Write C bus to register