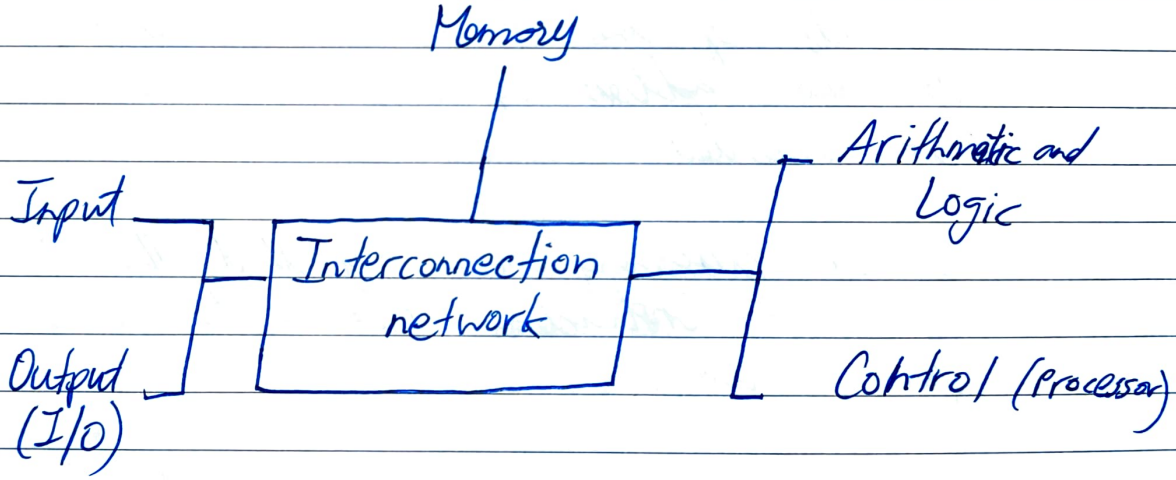
__/__/____

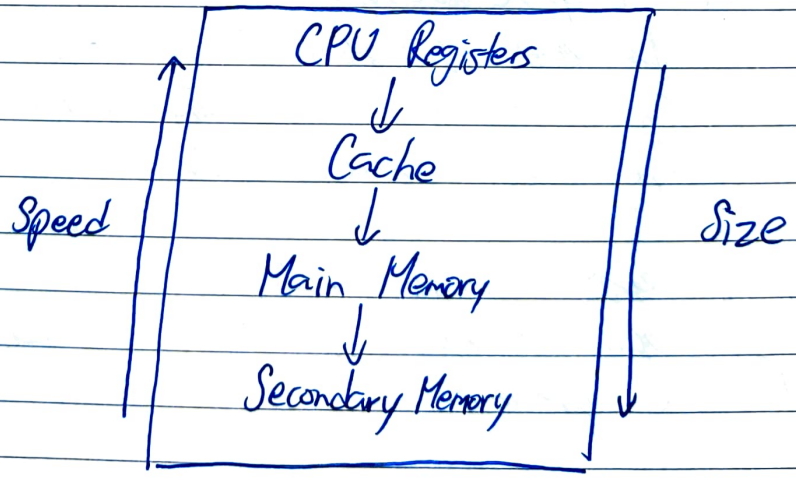
COA

Books

1. Carl Hamacher, Comp. Organization & Embedded Systems
2. William Stallings
3. Moh. Rafi. and Rajan Chandra, Modern Computer Arch.

Functional Units





__/__/____

- The instruction register (IR) holds the instruction that is currently being executed.
- The program counter (PC) is another specialized register:

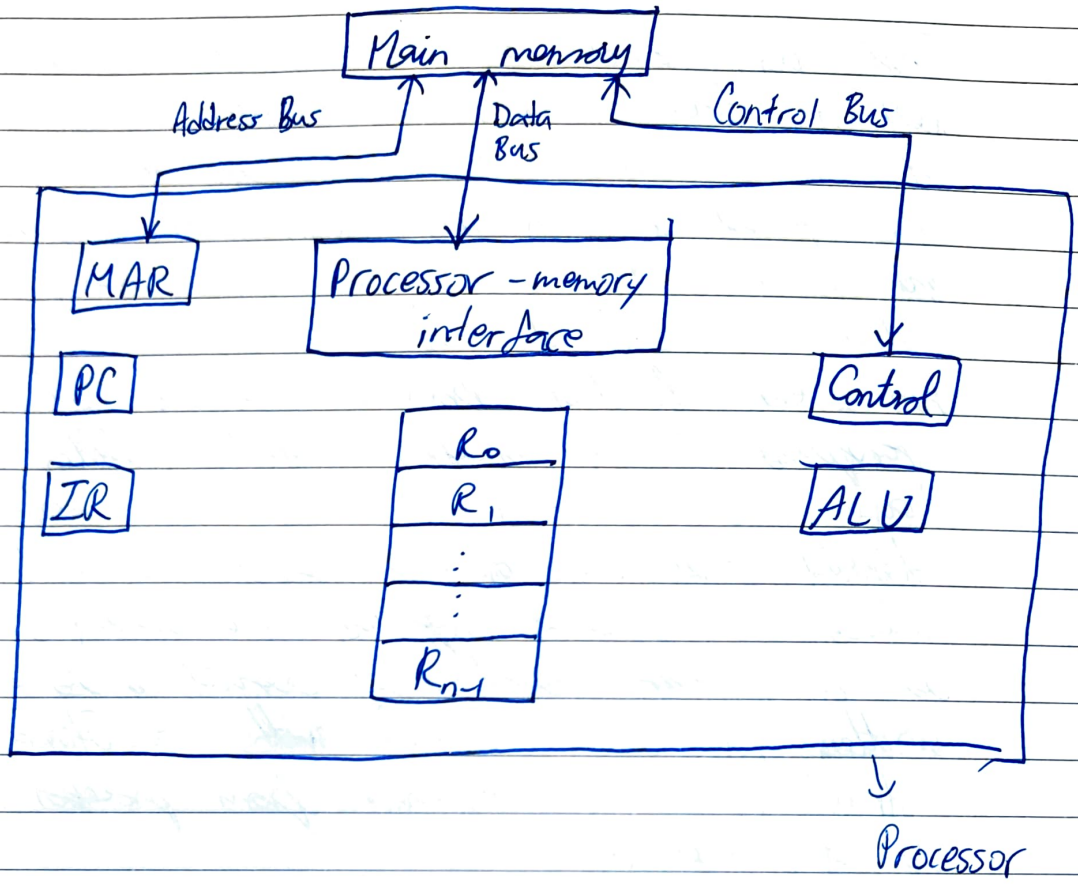
(i) It contains the memory address of the next instruction to be fetched and executed.

(ii) During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed.

(iii) It is customary to say that the PC points to next instruction that is to be fetched from memory.

- General-purpose registers ($R_0 - R_{n-1}$), often called processor registers. They serve a variety of functions including holding operands that have been loaded from memory for processing.
- The processor-memory interface is a circuit which manages the transfer of data between main memory and the processor.

__/__/____



- A program must be in the main memory in order for it to be executed. It is often transferred there from secondary storage through input unit
- Execution of program begins when PC is set to point to the first instruction of the program.
- The contents of the PC are transferred to the memory along with a Read control signal.
- When the addressed word has been fetched from the memory, it is loaded into register IR.
- At this point, the instruction is ready to be interpreted (CPU decodes and understands) and executed

- Instructions such as Load, Store, and Add perform data transfers and arithmetic operations. If an operand that resides in the memory is required for an instruction, it is fetched by sending it's address to the memory and initiating a Read operation. When the operand has been fetched from the memory, it is transferred to a processor register. After operands have been fetched, the ALU can perform a desired arithmetic operation such as Add on values in processor registers. The result is sent to a process register. If result is to be written into the memory with a Store instruction, it is transferred from processor register to the memory

- Some typical operating steps:

(i) Programs reside in the memory through input devices

(ii) PC is set to point to the first instruction

(iii) The contents of PC are transferred to MAR. A read signal is sent to memory.

(iv) The first instruction is read out and loaded into MDR.

(v) Contents of MDR are transferred to IR

(vi) Decode and execute the instruction. Get operands for ALU (Address to MAR - Read - MDR to ALU)

(vii) Perform operation in ALU and store the result back to general-purpose registers.

(viii) Transfer result to memory (address to MAR, result to MAR - Write)

(ix) During execution, PC is incremented to next instruction.

Q) List the steps needed to execute the given instruction

LOAD R2, LOC

→ • Send the address of the instruction word from register PC to the memory and issue a Read control command.

• Wait until the requested word has been retrieved from the memory.

• Load it into the register IR, where it is interpreted (decoded) by the control circuitry to determine the operation to be performed.

• Increment the contents of register PC to point to the next instruction in memory.

__/__/____

- Send the address value LOC from the instruction in register IR to the memory and issue a Read control command.
- Wait until the required word has been retrieved from the memory, then load it into register R2.

☆ Number Representation & Arithmetic Operations

- In computer systems, numbers are represented by a string of bits (0s and 1s) called a binary number.

Ex- $5 \rightarrow 0101 \rightarrow 1*2^0 + 0*2^1 + 1*2^2 + 0*2^3$

- In binary numbers, the MSB (most significant bit) on the leftmost side represents the sign of the number ($+ \rightarrow 0$, $- \rightarrow 1$)

- 1's compliment: $5 \rightarrow 0101$
  $5' \rightarrow 1010$
  (Also can be obtained by $(2^n - 1) - (\text{number})$)
  $n \rightarrow$ no. of bits used

- 2's compliment: $5'' \rightarrow 1010\ (5')$
  $+\ 1$
  1011
  (Also can be obtained by $(2^n) - (\text{number})$)

__/__/____

- Add/Subtract 1's Comp: Do not ignore carry, instead add 1 to end result.

Note: $-2 - 3 = (-2) + (-3)$

- Add/Subtract 2's Comp: Ignore carry in end result.

Note: To verify answer, find 2's compliment of (-)ve number to get (+)ve of same number.

- Using 2's compliment, n bits can represent values in the range of (-2^{n-1}) to $(2^{n-1} - 1)$. If the result is outside this range, then overflow has occurred.

- When both numbers to be added have same sign and are different from sign of result, overflow has occurred.

—x—

Instruction Set Architecture (ISA)

- A complete instruction set, including operand addressing methods, is often referred to as instruction set architecture (ISA) of a processor.
- To execute a high level language program on a processor, the program must be translated to the machine language for that processor, which is done by a compiler program.
- Assembly language is a readable symbolic representation of machine language.

Memory Locations and Addresses

- Memory consists of many storage cells each of which can store bit of info having value of 0 or 1.
- The usual approach is to deal with bits in groups, so that they can be together retrieved in a single basic operation.
- Each group of n bits is referred to as a word of information (n → word length).
- Memory → collection of words.
- 1 byte = 8 bits
- Modern computers have word lengths of 16 - 64 bits

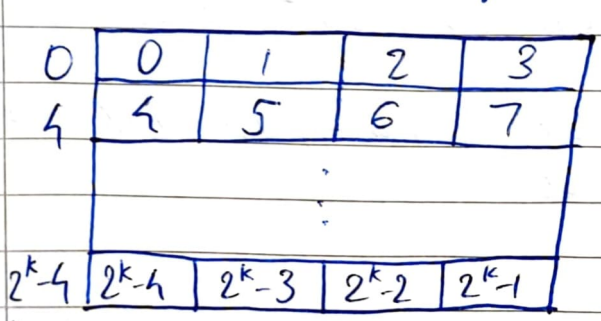
Ex:- If word length of computer is 32 bits.

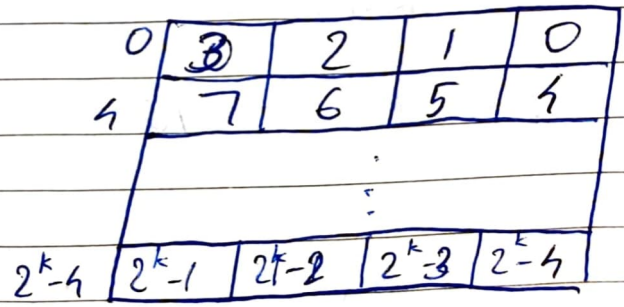
8 bits	8 bits	8 bits	8 bits
ASCII char	ASCII char	ASCII char	ASCII char

- It is customary to use numbers from 0 to $2^k - 1$ for a suitable value of k.
- Memory can have 2^k addressable locations.
- Byte locations have addresses 0, 1, 2, ...
  If word length of machine is 32 bits, successive words are located at addresses 0, 4, 8, ... with each word = 4 bytes

✱ Big-Endian & Little-Endian Assignments

- Big endian → lower byte addresses used for MSB of word (leftmost bytes)
- Little endian → lower byte addresses used for LSB of word (rightmost bytes)
- MSB → power of 2 is highest
  LSB → power of 2 is lowest.





__/__/____

- These are addresses used when accessing memory to store / retrieve a word.

☆ Instructions & Sequencing

- A computer must have instructions capable of performing four types of operations:

(i) Data transfers between memory and processor registers

(ii) Arithmetic and logic operations on data.

(iii) Program sequencing & control.

(iv) I/O transfers

☆ Register Transfer Notation (RTN)

- To describe transfer of information from one location in a computer to another.
- Addresses of memory locations → LOC, PLACE, A, VAR2
- Predefined names for processor registers → R_0, R_1, ... R_5
- Registers in I/O → DATAIN, OUTSTATUS

Ex: $R_2 \leftarrow [LOC]$ (contents of memory loc LOC → processor register R_2)

$R_4 \leftarrow [R_2] + [R_3]$

__/__/____

- LHS → Name of destination location
  RHS → Value

- Only contents of destination will change., not contents of source.

✱ Assembly Language Notation (ALN)

- Load R2, LOC (contents from memory location LOC transferred to processor register R_2)
- Add R_4, R_2, R_3 ($R_4 = R_2 + R_3$)

✱ RISC

- Computers that restrict each instruction into a single word in memory to reduce complexity, and include different types of instructions in the instruction set of are called Reduced Instruction Set Computers (RISC)

- Each instruction fits in a single word.

- Memory operands are accessed only using Load / Store

- Load destination, source
  (or)
  Load processor_register, memory_location

- Addressing modes are the ways in which memory location can be specified.

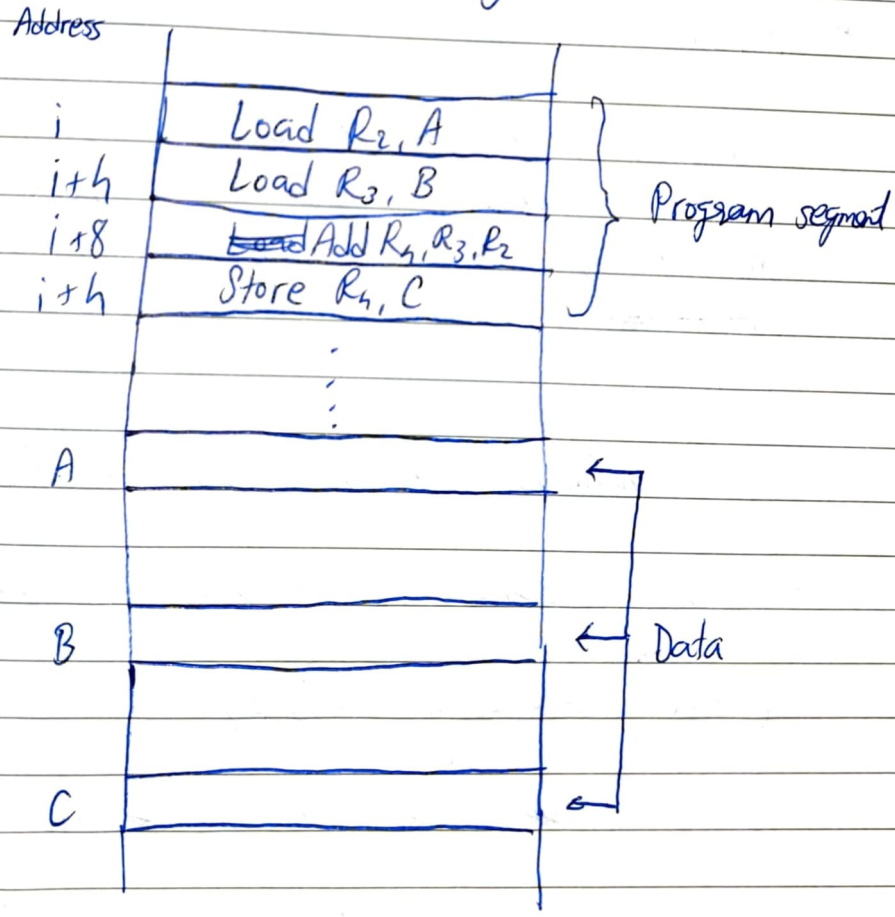
- Add, destination, source 1, source 2

__/__/____

Exc:- $C \leftarrow [A] + [B]$

Load R_2, A
Load R_3, B
~~Load R_4~~
Add R_4, R_2, R_3
Store R_4, C

- Store source, destination
(or)
Store processor_register, memory_location



- Processor contains a register called program counter (PC) which holds address of next instruction to be executed

- Before execution, address of first instruction (i) → PC
- Processor control circuits use information in PC to fetch and execute instructions, one by one, in increasing order (straight line sequencing).
- During execution of each instruction, PC + 4. → (size of instruction)
- There are two phases while executing a given instruction:

① Instruction fetch (Memory Location → IR (instruction register) (Address is in PC)

② Instruction Execute (Arithmetic / Logic operation → stored in destination location)

★ Branching

i	Load R_2, NUM1
i+4	Load R_3, NUM2
i+8	Add R_2, R_2, R_3
i+12	Load R_3, NUM3
i+16	Add R_2, R_2, R_3
⋮	⋮
i+8n-12	Load R_3, NUMn
i+8n-8	Add R_2, R_2, R_3
i+8n-4	Store R_2, SUM
⋮	⋮
SUM	
NUM1	
⋮	⋮
NUMn	

→

	Load R_2, N
i+4	Clear R_3
LOOP: i+8	Determine address of next number, load into R_5, add to R_3
	Subtract R_2, R_2, #1
	Branch_if_[R_2]>0 LOOP
	Store R_3, SUM
	⋮
SUM	
N	
NUM1	
⋮	⋮

__/__/____

- Here R_2 → number of numbers.
  R_5 → current number.
  R_3 → SUM

☆ Addressing Modes

- The different ways for specifying locations of instruction operands are known as addressing modes.

Name	Syntax	Function
Immediate	#Value	Operand = Value
Register	R_i	$EA = R_i$
Absolute	LOC	EA = LOC
Register indirect	(R_i)	$EA = [R_i]$
Index	$X(R_i)$	$EA = [R_i] + X$
Base with index	(R_i, R_j)	$EA = [R_i] + [R_j]$

- Here, EA = effective address
  X = index value.

- **Register Mode** : Operand is contents of processor register.
- **Absolute Mode** : Operand is in a memory location.

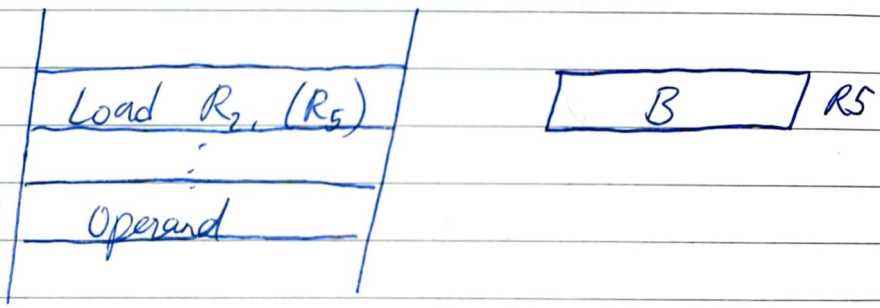
Ex:- Add R_4, R_2, R_3 (register mode)
Load R_2, NUM1 (absolute mode)

— Immediate mode

- Add $R_4, R_6, \#200$ (Adding value 200 to contents of R_6 and storing in R_4)
- Clear R_3 ($[R_3] \rightarrow 0$)

Note • In many RISC-type processors, one general-purpose register is dedicated to holding a constant value of 0. (register R_0), whose contents cannot be changed.

— • Indirect Mode: The effective address of operand is the contents of register specified in instruction.



- It allows the instruction to use a register that contains the address of operand rather than operand itself. Here, register acts as a pointer to a memory location.

Ex:-
```
        Load   R2, N
        Clear  R3
        Move   R4, #NUM1
LOOP:   Load   R5, (R4)          (Get next number)
        Add    R3, R3, R5        (Add to SUM)
        Add    R4, R4, #4        (Increment pointer to list)
        Subtract R2, R2, #1      (Decrement counter)
        Branch_if_[R2]>0 LOOP
        Store  R3, SUM
```

- Here, R_4 → pointer to numbers in the list.
- Since Load can only be used to load memory source operand, we use Move instead, thus fetching ~~contents of~~ location of NUM1 operand (address)
- Add $R_4, R_4, \#4$ is used to add contents of pointer R_4 to contain address of NUM2.

— **Index Mode** $(X(R_i))$

- $EA = X + [R_i]$

Ex:- Here, we want to add all the scores of a particular test by all n students and store them in SUM1, SUM2, SUM3

```
Move  R2, #LIST
Clear R3   (test1)
Clear R4   (test2)
Clear R5   (test3)
Load  R6, N   (counter)
LOOP: Load  R7, 4(R2)
~~Load~~
Add   R3, R3, R7
Load  R7, 8(R2)
Add   R4, R4, R7
Load  R7, 12(R2)
Add   R5, R5, R7
```

N	n
LIST	Student ID
LIST+4	Test 1
LIST+8	Test 2
LIST+12	Test 3
LIST+16	Student ID
	Test 1
	Test 2
	Test 3
	.
	.

S1 S2

```
Add     R2, R2, #16      (increment pointer/next student)
Subtract R6, R6, #1      (decrement counter)
Branch_if_[R6]>0 LOOP
Store   R3, SUM1         (Total of Test 1)
Store   R4, SUM2         (Total of Test 2)
Store   R5, SUM3         (Total of Test3)
```

✱ CISC

- It is a computer architecture in which single instructions can execute several low-level operations, which may span more than one word of memory.
- They are not constrained to load/store architecture.
- Instructions in CISC typically don't use 3-address format, mostly 2-address.

(Operation destination, source)

- Add B, A (CISC) ⇌ Add B, A, B (RISC)
- Add C, A, B (RISC) → Move C, B (CISC)
  Add C, A

(Move destination, source) (includes functionality of Load & Store)

- Move can be used between Memory ⇌ Processor Register
  Memory ⇌ Memory
  Register ⇌ Register

__/__/__

☆ Addressing Modes (CISC)

- Apart from (i) Immediate (ii) Register (iii) Absolute (iv) Indirect and (v) Index.

— **Autoincrement Mode** $(R_i)+$

- After accessing operand, contents of register are automatically incremented to point to the next operand in memory.
- Increment by 1 (byte-sized operands)
  2 (16-bit)
  4 (32-bit)

— **Autodecrement Mode** $-(R_i)$

- Here, contents of register are decremented before being used as the effective address (EA)

Note:
- To push a new item on the stack

  Subtract SP, #4 ⟶ Move -(SP), NEWITEM (letter)
  Move (SP), NEWITEM

- To pop an item from stack

  MOVE ITEM, (SP) ⟶ Move ITEM, (SP)+
  Add SP, #4

- ## Relative Mode

- EA is determined by index mode using program counter (PC) instead of R_i : $[X(PC)]$
- Addressed location is identified relative to the PC.

Condition Codes

- Operations performed by processor typically generate results such as numbers (+, −, 0)
- Processor can maintain the information by recording in individual bits called flags, which are grouped together in a special processor register called condition code register.

(i) N (negative) (T: 1, F: 0)
(ii) Z (zero) (T: 1, F: 0)
(iii) V (overflow) (T: 1, F: 0)
(iv) C (carry) (T: 1, F: 0) (on the MSB)

- Overflow occurs when result of arithmetic operation is outside range of values: (-2^{n-1}) to $(2^{n-1}-1)$ for n bits.

Ex:- For the Subtract instruction for register R_2,

Branch_if_[R_2] > 0 LOOP

↓

Branch > 0 LOOP

Ex:

```
        Move   R2, N
        Clear  R3
        Move   R4, #NUM1
LOOP:   Add    R3, (R4)+
        Subtract R2, #1
        Branch >0 LOOP
        Move   SUM, R3
```

RISC	CISC
• Reduced Instruction Set Computer	• Complex Instruction Set Computer.
• These chips are relatively simple to design	• These chips are complex to design.
• Inexpensive.	• Relatively expensive
• Less number of instructions	• More number of instructions
• Does not support arrays	• Supports arrays
• Registers are used for procedure arguments and return addresses	• Stack is used for procedure arguments and return addresses
• It is a microprocessor architecture that uses a small instruction set of uniform length.	• Offers hundreds of instructions of different sizes to the users.
Ex:- SPARC, POWER PC	Ex:- Intel, AMD

A(1,2,3)
B(4,5,6)

$A \cdot B = (1\times4) + (2\times5) + (3\times6) = 32$

__/__/____

☆ Examples

① Vector Dot Product

RISC

```
        Move    R2, #A          (R2 points to vector A)
        Move    R3, #B          (R3 points to vector B)
        Load    R4, N           (counter)
        Clear   R5              (dot product)
LOOP:   Load    R6, (R2)
        Load    R7, (R3)
        Multiply R8, R6, R7     (corresponding elements)
        Add     R5, R5, R8
        Add     R2, R2, #4      (increment pointer)
        Add     R3, R3, #4
        Subtract R4, R4, #1     (decrement counter)
        Branch_if_[R4]>0 LOOP
        Store   R5, DOTPROD
```

CISC

```
        Move    R2, #A
        Move    R3, #B
        Move    R4, N
        Clear   R5
LOOP:   Move    R6, (R2)+
        Multiply R6, (R3)+
        Add     R5, R6
        Subtract R4, #1
        Branch >0 LOOP
        Move    DOTPROD, R5
```

~~0111~~ + 1001 / 1010 ; 0111 + 1 / 1000 ; 0111

__/__/____

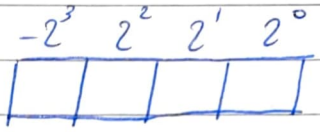
ARITHMETIC & LOGIC UNIT

★ Addition & Subtraction

- For unsigned numbers, carry indicates overflow.
- For signed numbers, overflow is indicated when:

(i) When two (+)ve numbers are added and result is (+)ve (Carry is considered)

(ii) When two (+)ve numbers are added and result is (−)ve



(2's complement)

0011	(3)	1100	(−4)
+0100	(4)	+0100	(4)
0111	(7)	10000	(0)

0101	(5)	1001	(−7)
+0100	(4)	+1010	(−6)
1001	(−7)?	10011	(Overflow)
Overflow		$(-2^4+2^1+2^0)=(-13)$	

Ex: ~~0010~~ ~~(−7)~~

M = 5 = 0101

S = −2 = 1110

2 = 0010

M + S'' = M − S = 0101 + 0010 = 0111 = 7 //

$$\begin{array}{r} 1011 \\ +\ \ 1 \\ \hline 1100 \end{array}$$

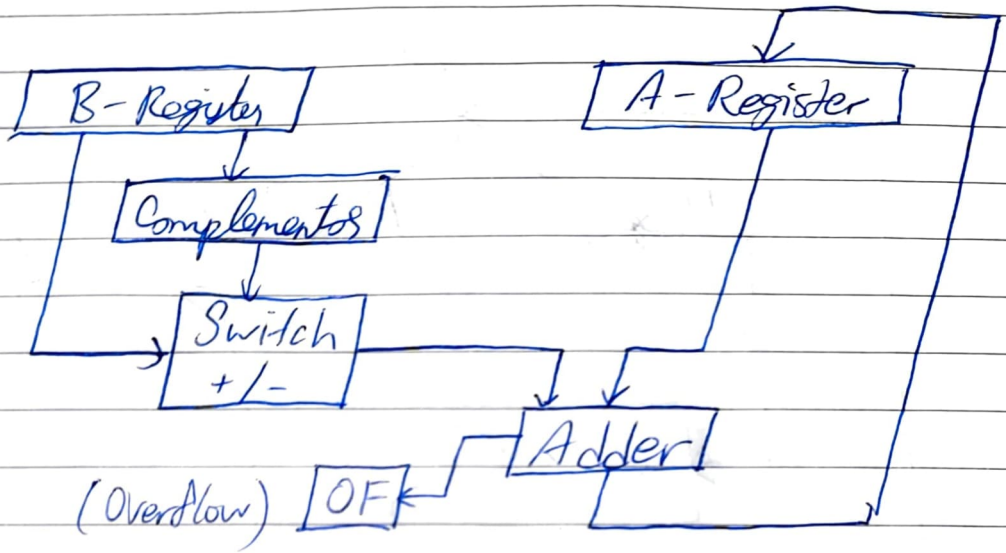
__/__/__

Ex: $M = -6 = 1010$ $\quad$ $\begin{array}{r} 1010 \\ +\ 1100 \\ \hline 10110 \end{array}$ (Overflow)

$S = 4 = 0100$

$-4 = 1100$

$M - S = M + S'' = -10$ $\quad$ $-2^4 + 2^2 + 2^1 = -10$



★ Multiplication

- Multiplication of two n-bit binary integers results in a product of up to 2n bits in length.

$$\begin{array}{r} 1011 \\ \times\ 1101 \\ \hline 1011 \\ 0000\times \\ 1011\times\times \\ 1011\times\times\times \\ \hline 10001111 \end{array}$$

Reverse

$1011 \times 1 \times 2^0$

$1011 \times 0 \times 2^1$

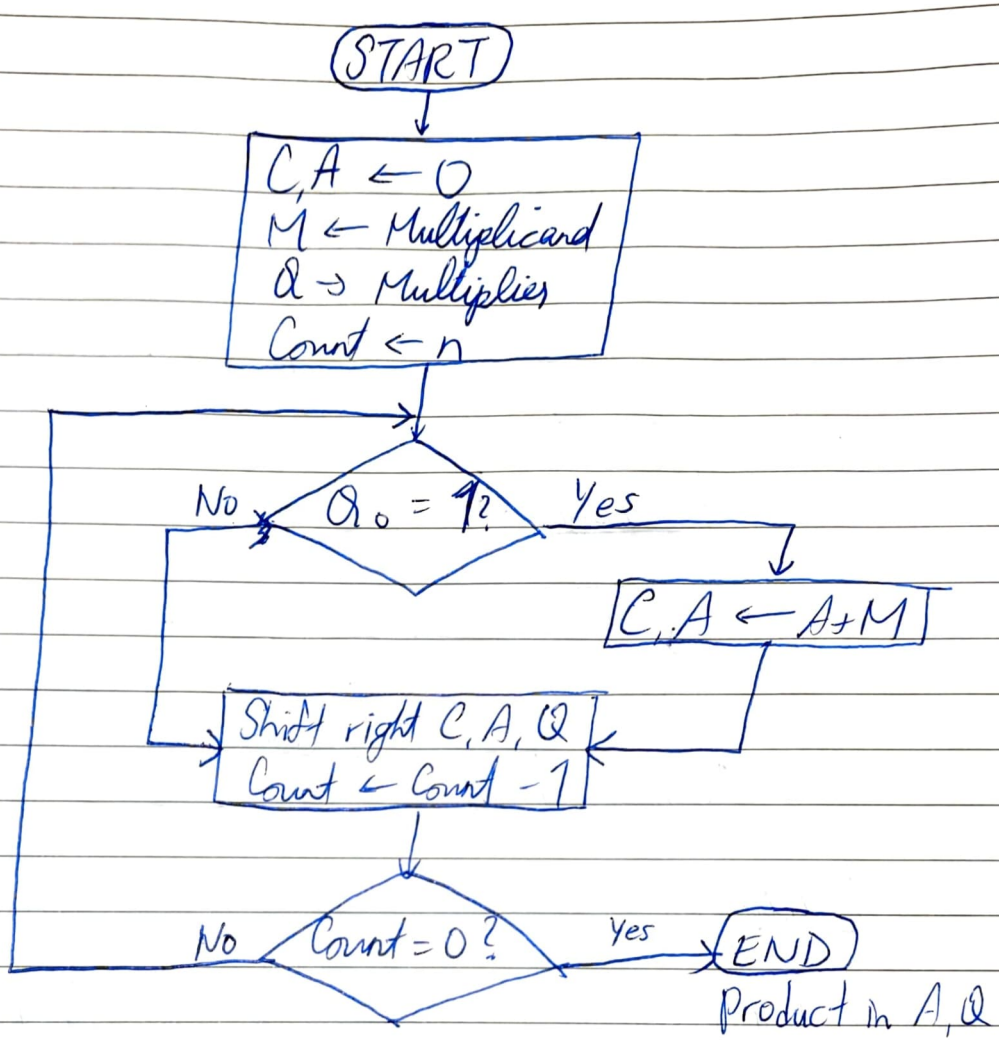
$1011 \times 1 \times 2^2$

$1011 \times 1 \times 2^3$

} Partial Products

$$\begin{array}{r} 0010 \\ +1011 \\ \hline 1101 \end{array} \qquad \begin{array}{r} 0110 \\ +1011 \\ \hline 10001 \end{array}$$

__/__/____



Ex: $M = 1011^{(11)}$, $M'' = 0101$

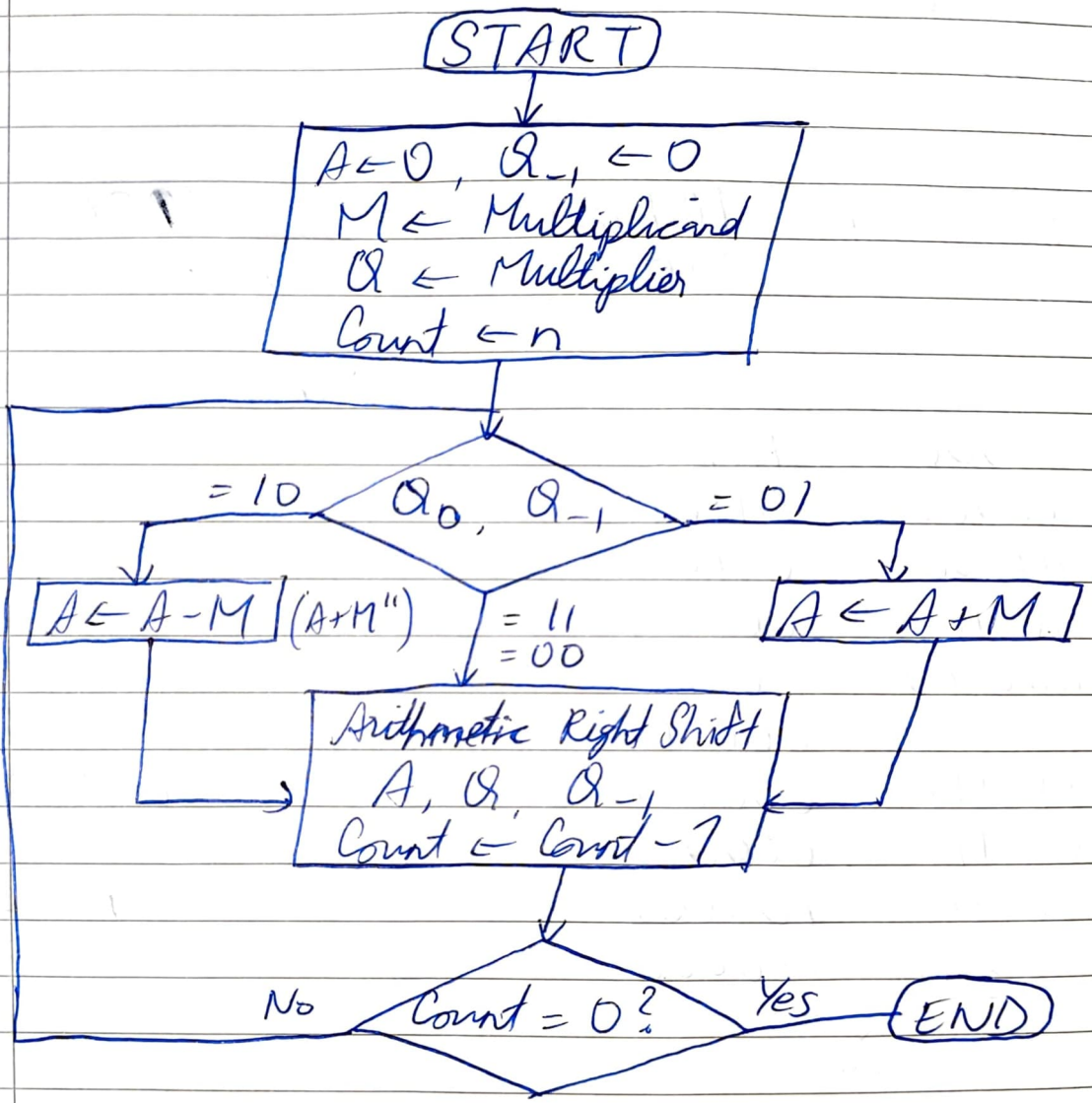
$Q = 1101^{(13)}$, $n = 4$

	C	A	Q (q_0)	M	
	0	0000	1101	1011	
A+M	0	1011	1101	1011	
>>	0	0101	1110	1011	-①
>>	0	0010	1111	1011	-②
A+M	0	1101	1111	1011	
>>	0	0110	1111	1011	-③
A+M	1	0001	1111	1011	
>>	0	1000	1111	1011	-④

143

☆ Booth's Algorithm

- Useful for multiplying negative numbers using 2's complement.



Ex:- $M = 0111 = 7$, $M'' = 1001$

$Q = 0011 = 3$, $n = 4$

	A	Q	Q-1	M
	0000	0011	0	0111
A-M	1001	0011	0	0111
>>	1100	1001	1	0111

→ ①

0111 + 1110 = 10101

01011 + 1 = 01100

000110 + 101000 = 101110

111011 + 000111 = 100001

__/__/____

	A	Q	Q_{-1}	M	
>>	1110	0100	1	0111	-①
A+M	0101	0100	1	0111	
>>	0010	1010	0	0111	-③
>>	0001	0101	0	0111	-④

21

Ex: $M = -24 = 101000$, $M'' = 011000$

$Q = -10 = 110110$ $\quad n = 6$

	A	Q	Q_{-1}	M	
	000000	110110	0	101000	
>>	000000	011011	0	101000	-①
A-M	011000	011011	0	101000	
>>	001100	001101	1	101000	-②
>>	000110	000110	1	101000	-③
A+M	101110	000110	1	101000	
>>	110111	000011	0	101000	-④
A-M	001111	000011	0	101000	
>>	000111	100001	1	101000	-⑤
>>	000011	110000	1	101000	-⑥

+240

Ex: $M = -25 = 100111$, $M'' = 011001$

$Q = -4 = 111100$

	A	Q	Q_{-1}	M	
	000000	111100	0	100111	
>>	000000	011110	0	100111	-①
>>	000000	001111	0	100111	-②
A-M	011001	001111	0	100111	

$$\begin{array}{r} 000110 \\ +100111 \\ \hline 101101 \end{array}$$

$$1011 \rightarrow 0101$$

$$\begin{array}{r} 0111 \\ +0101 \\ \hline 1100 \end{array}$$

$$\begin{array}{r} 10010 \\ +0101 \\ \hline 10111 \end{array}$$

$$\begin{array}{r} 1110 \\ +0101 \\ \hline 10011 \end{array}$$

__/__/____

	A	Q	Q_{-1}	M	
>>	001100	100111	1	100111	-③
>>	000110	010011	1	100111	-④
>>	000011	001001	1	100111	-⑤
>>	000001	100100	1	100111	-⑥

100

Ex: $M = -25 = 100111$, $M'' = 011001$

$Q = 3 = 000011$

	A	Q	Q_{-1}	M	
	000000	000011	0	100111	
A-M	011001	000011	0	100111	
>>	001100	100001	1	100111	-①
>>	000110	010000	1	100111	-②
A+M	101101	010000	1	100111	
>>	110110	101000	0	100111	-③
>>	111011	010100	0	100111	-④
>>	111101	101010	0	100111	-⑤
>>	111110	110101	0	100111	-⑥

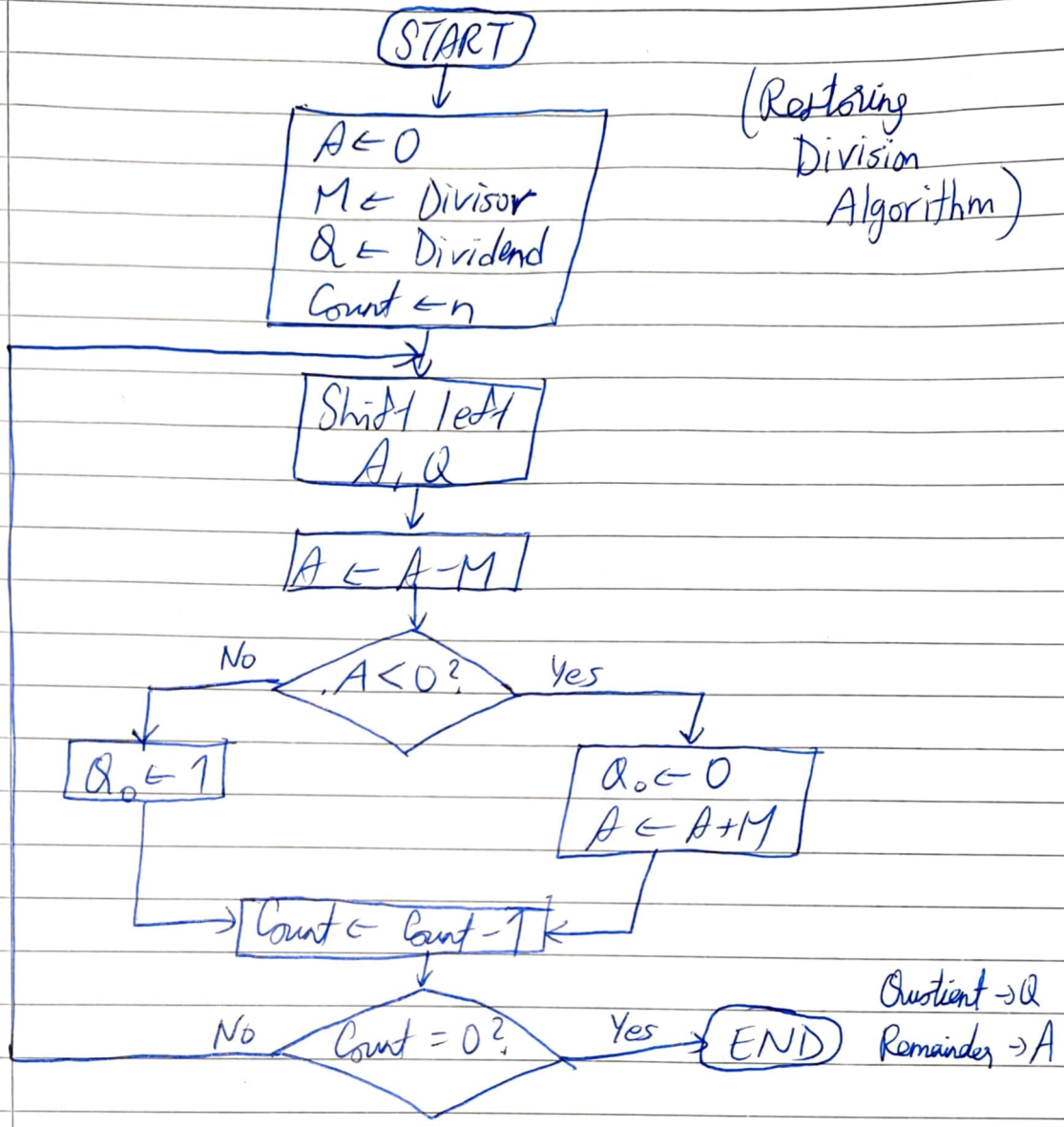
Division

$$\begin{array}{r} 00001101 \\ 1011 \overline{)\,10010011} \\ 1011\downarrow \\ \hline 01110 \\ 1011\downarrow \\ \hline 00111 \\ 1011\downarrow \\ \hline 1100 \end{array}$$

$147 \div 11 = \sim 13$

```
 1111
 01011
+11101
11 1000
```

__/__/____



(Restoring Division Algorithm)

Ex: 8 / 3 = n = 5

Q = 01000

M = 3 = 00011 , M'' = 11101

	A	Q	
	00000	01000	
<<	00000	1000_	
A-M	11101	10001	-①
<<	11011	0001_	
A-M	11000	00011	-②

$3\overline{)\frac{8}{2}}^{2}$

$10000 + 11101 = 101101$

$010010 \to 00100 + 11101 = 100001$

$11101 + 00011 = 100000$, $11111 + 00011 = 100010$

$11101 + 11110 = 00010 + 00011 = 00101$

$11110 + 00011 = 100001$, $11101 + 00010 = 11111 + 00011 = 100010$

$00010 + 11101 = 11111$

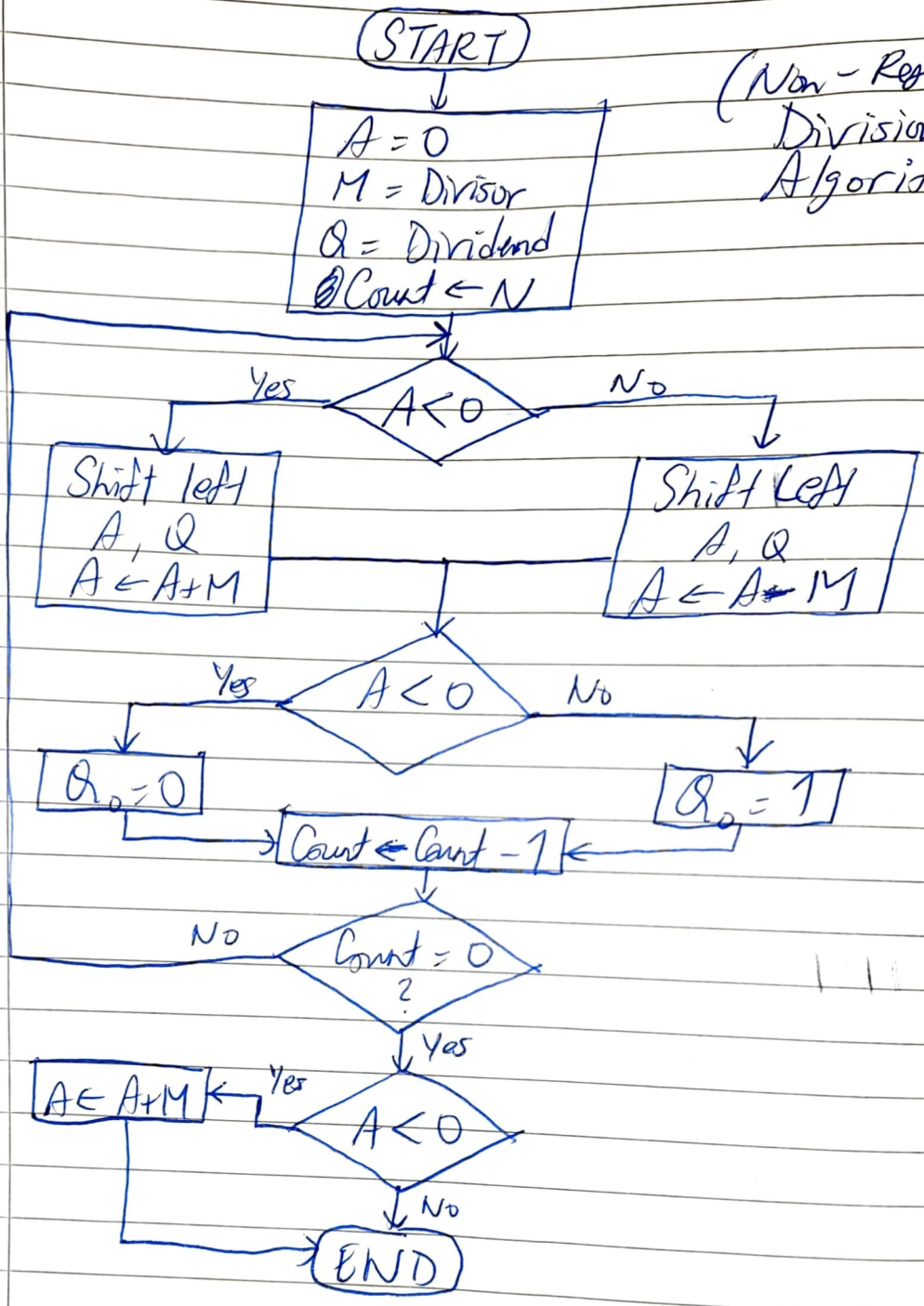
	A	Q
<<	X0000	0011_
A-M	101101	

	A	Q	
	00000	01000	
<<	00000	1000_	
A-M	11101	1000 0	
A+M	00000	10000	-①
<<	00000	0000_	
A-M	11110	00000	
A+M	00001	00000	-②
<<	00010	0000_	
A-M	11111	00000	
A+M	00010	00000	-③
<<	00100	0000_	
A-M	00001	00001	-④
<<	00010	0001_	
A-M	10101	00010	
A+M	00010	00010	-⑤
	2	2	

- $D = Q \times d + R$
- $\text{sign}(R) = \text{sign}(D)$
- $\text{sign}(Q) = \text{sign}(D) \times \text{sign}(d)$

__/__/____

(Non - Restoring Division Algorithm)



Ex: $Q = 11 = 1011$

$M = 3 = 0011$, $M'' = 1101$, $n = 4$

PTO →

$\begin{array}{r} 1101 \\ +0001 \\ \hline 1110 \end{array}$ $\begin{array}{r} 1100 \\ +0011 \\ \hline 1111 \end{array}$ $\begin{array}{r} 1111 \\ +0011 \\ \hline 10010 \end{array}$ $\begin{array}{r} 0101 \\ +1101 \\ \hline 10010 \end{array}$ $3\overline{)11}\ \begin{array}{r} 3 \\ 9 \\ \hline 2 \end{array}$

__/__/____

	A	Q	
	0000	1011	
<<	0001	011_	
A-M	1110	0110	-①
<<	1100	110_	
A+M	1111	1100	-②
<<	1111	100_	
A+M	0010	1001	-③
<<	0101	001_	
A-M	0010	0011	-④
	2	3	

☆ Floating Point Numbers

Exo: Convert decimal → binary : 4.25

4 → 100

0.25 → 0.25 × 2 = 0.5 → 0 ↓

0.5 × 2 = 1 → 1 ↓

∴ $(4.25)_{10} = (100.01)_2$

Exo: Convert binary → decimal : 10100.101

1	0	1	0	0	.	1	0	1
2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}

$= 20.625$

Exi: 9.76×10^{14}

$\pm S \times B^{\pm E}$

S → significand

B → base (implicit)

E → exponent

__/__/____

32-bit IEEE Format

- Sign bit
- 23 significand bit (mantissa)
- 8 bits for signed exponent.

$$\pm 1.M \times 2^{E'-127}$$

S	E'	M
(sign)	(8-bit exponent) (excess 127)	(mantissa) (23-bits)

Ex: 0 | 10000001 | 01000110011001100110011

S = 0 (+)

$E' = (10000001)_2 = (129)_{10}$

$E = 129 - 127 = 2$

Normalized form: $1.01000110\ldots \times 2^2$

$= 101.0001100\ldots$

$= 5$

//

Ex: 1 | 01111111 | 1000000....

S = 1 (−)

E' = 127

E = 0

$N.F = -1.100\ldots \times 2^0$

$= -1.1$

$= -1.5$

//

__/__/____

- From decimal → 32-bit.

Ex:- 22.125

S = 0 (+)

22 = 10110

0.125 × 2 = 0.25 → 0

0.25 × 2 = 0.5 → 0

0.5 × 2 = 1 → 1

} 22.125 = 10110.001

$22.125 = 1.0110001 \times 2^4$

E = 4 , E' = 127 + 4 = 131 = 10000011

∴ 0 | 10000011 | 01100010....

☆ Floating Point Arithmetic

- Exponent overflow : (+) exponent exceeds maximum possible exponent value $(-\infty)(\infty)$
- Exponent underflow : (−) exponent less than minimum possible exponent value $(-200 < -127)$ (0)
- Significand underflow : Digits may flow off right end of significand.
- Significand overflow : Addition of two significands of same sign may result in carry out in MSB.
- Divide by zero
- Invalid operations such as 0/0 or $\sqrt{-1}$ are attempted.

$\frac{1}{2} + \frac{1}{8} = \frac{5}{8}$ $\frac{5}{8}$

__/__/__

Addition / Subtraction of Floating Point Numbers

- Choose no. with smallest exponent and shift it's mantissa right a no. of steps = diff in exponents.
- Set exponent of result equal to larger exponent.
- Perform addition / subtraction on mantissas and determine sign. Normalize if required (1.M format)

Ex: $A = 96.625$ (+) $B = 12.125$

$96 = 01100000$
$0.625 \times 2 = 1.25 \rightarrow 1$
$0.25 \times 2 = 0.5 \rightarrow 0$
$0.5 \times 2 = 1 \rightarrow 1$

$12 = 1100$
$0.125 \times 2 = 0.25 \rightarrow 0$
$0.25 \times 2 = 0.5 \rightarrow 0$
$0.5 \times 2 = 1 \rightarrow 1$

$A = 1100000.101$
$= 1.100000101 \times 2^6$

$B = 1100.001$
$= 1.100001 \times 2^3$
(smaller exponent)

$$\begin{array}{r} 1.100000101 \times 2^6 \\ + \quad 0.001100000\,1 \times 2^6 \\ \hline 1.101100100 \times 2^6 \end{array}$$

$= 1101100.100$
$= 108.75$ //

~~1 | 10110011 0~~
1 | 10000101 | 10110011 0000

Multiplication

- If either operand is 0, result = 0.
- Add exponents and multiply the mantissas and determine sign of result.
- Normalize if necessary.

__/__/__

Division

- If divisor = 0, result = ∞ / error (can't divide by 0)
  If dividend = 0, result = 0

- Subtract exponents and divide mantissas and determine sign of result. Normalize if required.

Ex: $X = 2.0 = 01 = 1.00 \times 2^1$ $\quad Y = 3.0 = 11 = 1.10 \times 2^1$

$X \times Y =$

$$\begin{array}{r} 1.10 \\ \times\ 1.00 \\ \hline 000 \\ 000x \\ 110xx \\ \hline 1.1000 \end{array} \times 2^{2} = 110.00 \Rightarrow 6$$

Ex: $A = 0\ |\ 10000100\ |\ 0100$
$B = 1\ |\ 00111100\ |\ 1100$

A: $S = 0$ (+)
$E' = 132$
$E = 7$
$A = 1.0100 \times 2^7$

B: $S = 1$ (−)
$E' = 60$
$E = 67$
$B = 1.$

```
  01111010
+         1
  01111011
+ 10001000
  11000011
```

__/__/____

Ex: $A = 96.625$ $\times$ $B = 12.125$

$= 1100000.101$ $= 1100.001$

$= 1.100000101 \times 2^6$ $= 1.100001 \times 2^3$

Adding the exponents $\rightarrow 2^9$

Multiplying mantissa's $\rightarrow 1.00100100111100101 \times 2^{10}$

$= 1171.578$

Ex: $X = 0.832 \times 10^3$ $(\div)$ $Y = 0.521 \times 10^3$

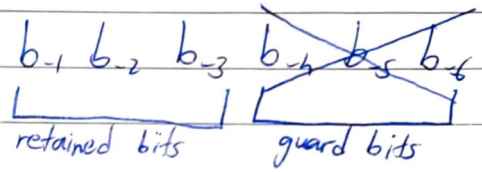
Exponent $\rightarrow 2^0$

$X = 0.832 \times 2 = 1.664 \rightarrow 1$ $Y = 0.521 \times 2 = 1.042 \rightarrow 1$

– Guard Bits

- Mantissas of initial operands and final results are limited to 24 bits (including implicit leading 1)
- Retaining extra bits, often called guard bits, yield maximum accuracy in final results.

(i) • One way to remove the guard bits is by chopping



for fractions in the range of $0.b_{-1}b_{-2}b_{-3}000 (\text{to}) 0.b_{-1}b_{-2}b_{-3}111$

(ii) • Another way is von Neumann rounding,
If bits to be removed are all 0's, $0.b_{-1}b_{-2}b_{-3}$ ~~000~~
If bits (any) to be removed are 1,
LSB of retained bits → 1.

If guard bits ≠ 000, then $0.b_{-1}b_{-2}1$

(iii) • Another way is rounding;
If there is 1 in the MSB of guard bits,
1 is added to LSB position of retained bits.

- $0.b_{-1}b_{-2}b_{-3}1 \longrightarrow 0b_{-1}b_{-2}b_{-3} + 0.001$
  $0.b_{-1}b_{-2}b_{-3}0 \longrightarrow 0.b_{-1}b_{-2}b_{-3}$

__/__/____

CONTROL UNIT

CPU

- **Processing section** (includes hardware elements ALU, shift registers, comparators, MUX's to operate on data)
- **Control Unit** (controls system operation by routing selected data items to selected processing hardware at right time)

- To initiate an operation within the system, signals are generated by the control unit, which are synchronized by a master clock.
- Inputs of Control Unit are (i) master clock (ii) status (condition) info from processing section (iii) command signals from external agent.
- Outputs produced are signals that drive the processing section and response to external environment (operation complete / aborted) due to exceptions (overflow / underflow)
- The key responsibilities of control unit are:

(i) Interpretation: Control unit reads instructions from memory (using PC as pointer), recognizes instruction type, gets operands and routes them to appropriate functional units of execution unit.

(ii) Sequencing : Control unit determines address of next instruction to be executed and ⟶ PC.

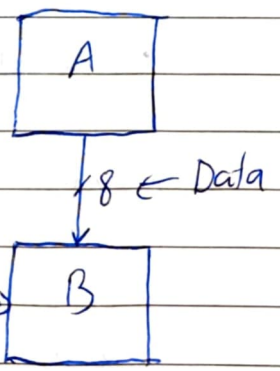
- A digital processing system is a collection of registers where processing activity is performed by a sequence of data transfer operations among registers either directly or with processing hardware (ALU)

Exi- 8-bit info moved from register A to register B.
⇒ B ← A , declare register A[8], B[8]

Exi- If higher order byte of 16-bit PC = one 8-bit register
Declare registers : PC[16]
Declare subregister : PCH1[8] = PC[15-8]

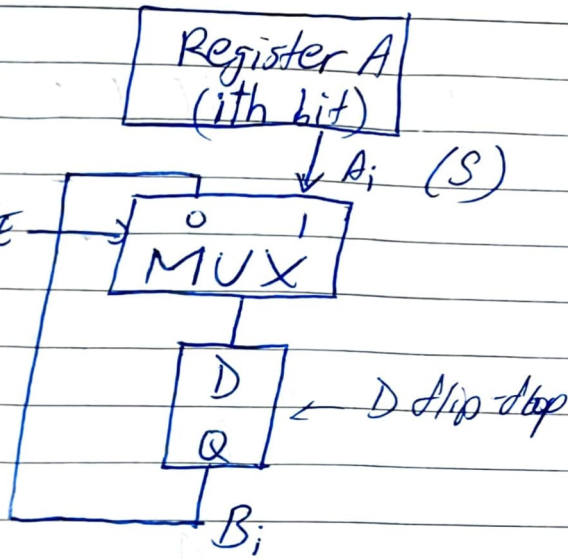
Exi- MSB of A copied to LSB of B
B[0] ← A[7]

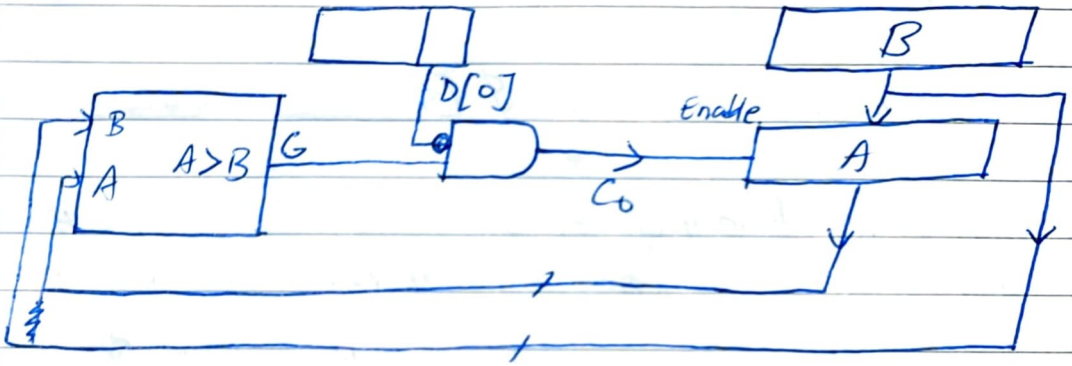
- Normally information transfer is controlled by enable signal E.
- For 16-bit PC, PCH1[8] = PC[15-8]
  PCL[8] = PC[7-0]
- Here E : B ← A
- Purpose of enable input is to make sure data transfer between A and B register takes place only under predetermined condition and not every clock pulse.



- $E=0 \Rightarrow$ Register B retains current value.
  $E=1 \Rightarrow B_i \leftarrow A_i$

- D flip-flop stored value chosen by MUX and updates value of B_i on clock edge.

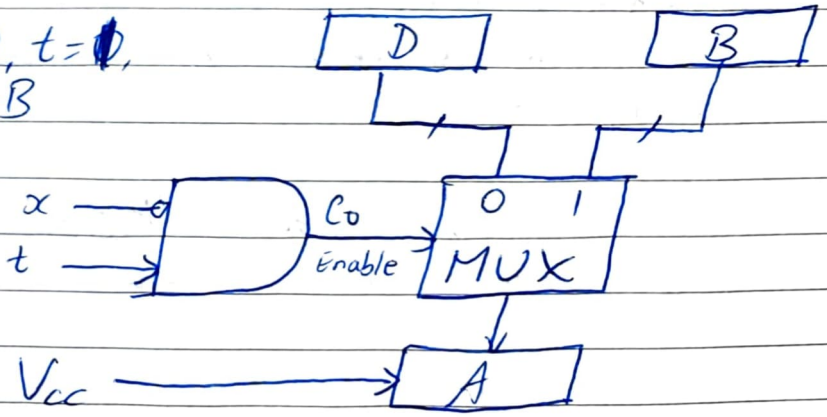




- Here, if $A > B$ and $D[0] = 0$, then $A \leftarrow B$

  C_0: $A \leftarrow B$ where $C_0 = G \& (D[0])'$

- Here, if $x = 0, t = 1$,
  $A \leftarrow B$
  else, $A \leftarrow D$



- C_0 : $A \leftarrow B$ (1) where $C_0 = x't$
  C_0' : $A \leftarrow D$ (0) where $C_0' = (x't)' = (x + t')$

__/__/____

Note $A = 10101010$

$ASR(A) = 11010101$ (arithmetic right shift)

$LSR(A) = 01010101$ (logical right shift)

Ex: (Register transfer notations)

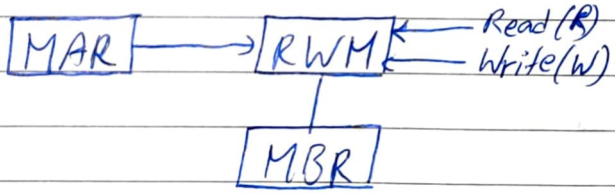
$D \leftarrow A'$ (transfer complement of A to D)

$A' + 1$ (2's complement of A)

$D \leftarrow A \vee B$ (OR gate)

$D \leftarrow A \wedge B$ (AND gate)

- For concatenation use $A \$ B$
- Read / Write Memory (RWM) is a part of the processing section where each word is considered as an addressable register



- Memory Address Register (MAR) is used as a pointer to a memory word, which holds the address of desired memory word.
- Memory Buffer Register (MBR) is used to temporarily store data being transferred.

$R: MBR \leftarrow M((MAR))$

$W: M((MAR)) \leftarrow MBR$

- M is used to indicate a memory. $M((MAR))$ indicates memory word X addressed by content of MAR.

__/__/____

Bus

- Buses are shared communication systems that transfer data between components inside a computer or between computers; consisting of multiple wires/lines that can carry data signals, info, etc.

- They are used to route data in and out of digital processing system. (in-bus and out-bus)

Eg:
Declare registers A[8], M[8], Q[8];
Declare buses inbus[8], outbus[8];
Start: A ← 0, M ← inbus;
Q ← inbus;
Loop: A ← A+M; Q ← Q-1;
If Q<0 (or) Q>0 then Loop
Outbus = A;
Halt: Go to Halt

M = 2 = 10 × Q = 3 = 11 (unsigned multiplication)

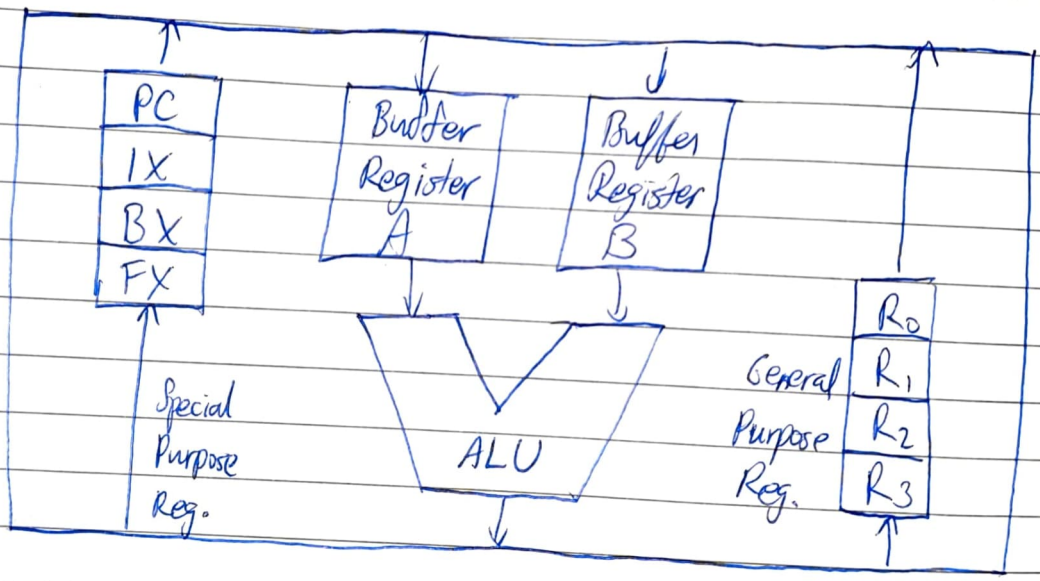
A	Q	Operation
00	11	
10	10	A ← A+M, Q = Q-1
100	01	A ← A+M, Q = Q-1
110 = 6 //	00	A ← A+M, Q = Q-1

- Comma inserted between operations that can be executed simultaneously, while semicolon between operations that can be executed serially.

__/__/____

Single Bus RALU

- All CPU registers are connected to the same bus. Whenever the ALU requires two operands, operands can be transferred only one at a time.
- Bus must be shared among various operands
- ALU must have buffer registers to hold transferred operand



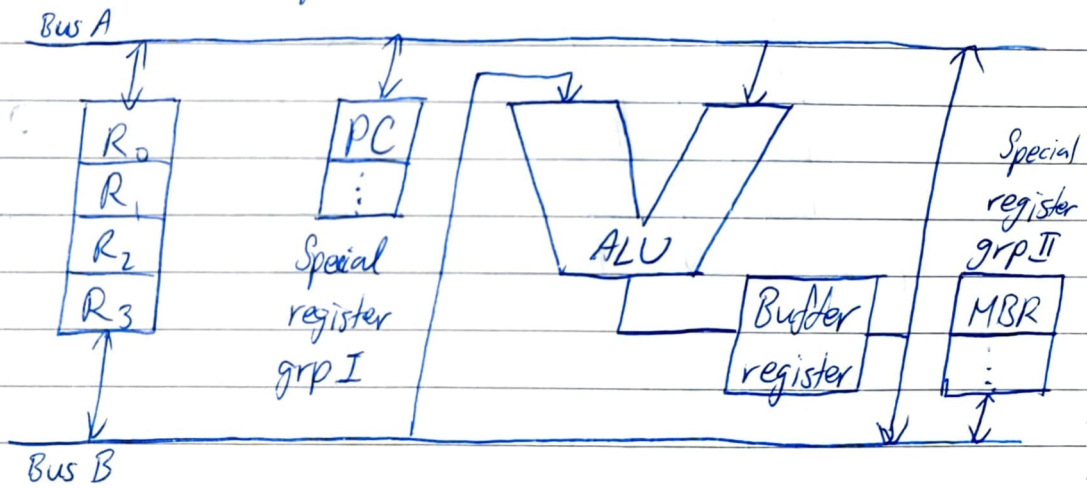
Ex:- For $R_2 \leftarrow R_0 + R_1$, it is completed in 3 clock cycles;

(i) Buffer A $\leftarrow [R_0]$
(ii) Buffer B $\leftarrow [R_1]$
(iii) Sum from ALU $\longrightarrow R_2$

- For single bus, if operands are in memory, two clock cycles are required to retrieve them.
- Speed ↓, Hardware ↑

Two Bus RALU

- All general-purpose registers are connected to both buses A and B, hence the two operands required by ALU are routed in one clock cycle.
- Information flowing on bus may be from general-purpose registers or special-purpose registers (divided into two groups).
- Data from two special-purpose registers of same group cannot be transferred to ALU at same time.



- Output of ALU may be routed to either special purpose registers or general-purpose registers.
- Transfer of required operands and loading of ALU output buffer register takes place in one cycle, then routed to required destination in second cycle.

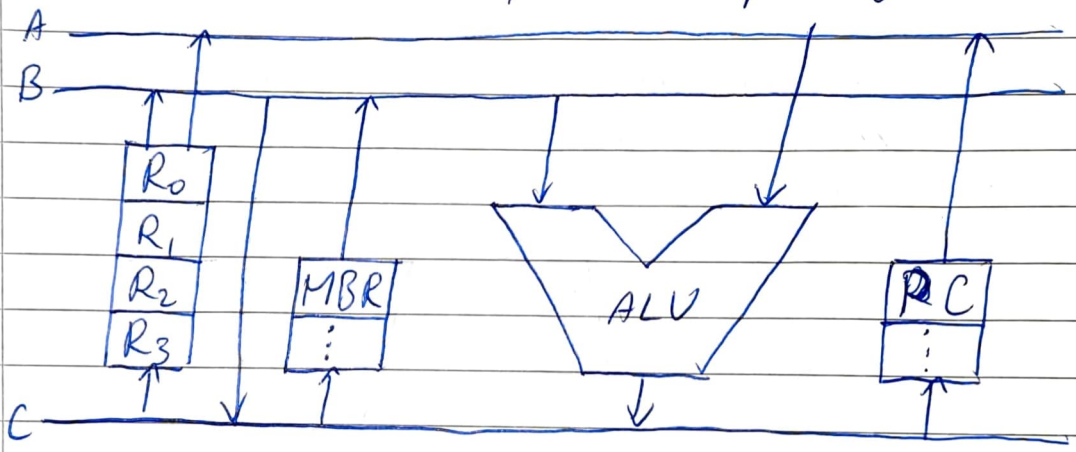
Ex: $R_2 \leftarrow R_0 + R_1$ can be completed in <u>two</u> clock cycles.

__/__/__

Three Bus RALU

- Addition of another bus C allows the system to perform operations such as $R_2 \leftarrow R_0 + R_1$ in one clock cycle but will increase system cost.

- Finite delay will occur due to transfer of operand to ALU and time taken by ALU for producing result.



Note

Speed of Operation $\Rightarrow$ $1 < 2 < 3$

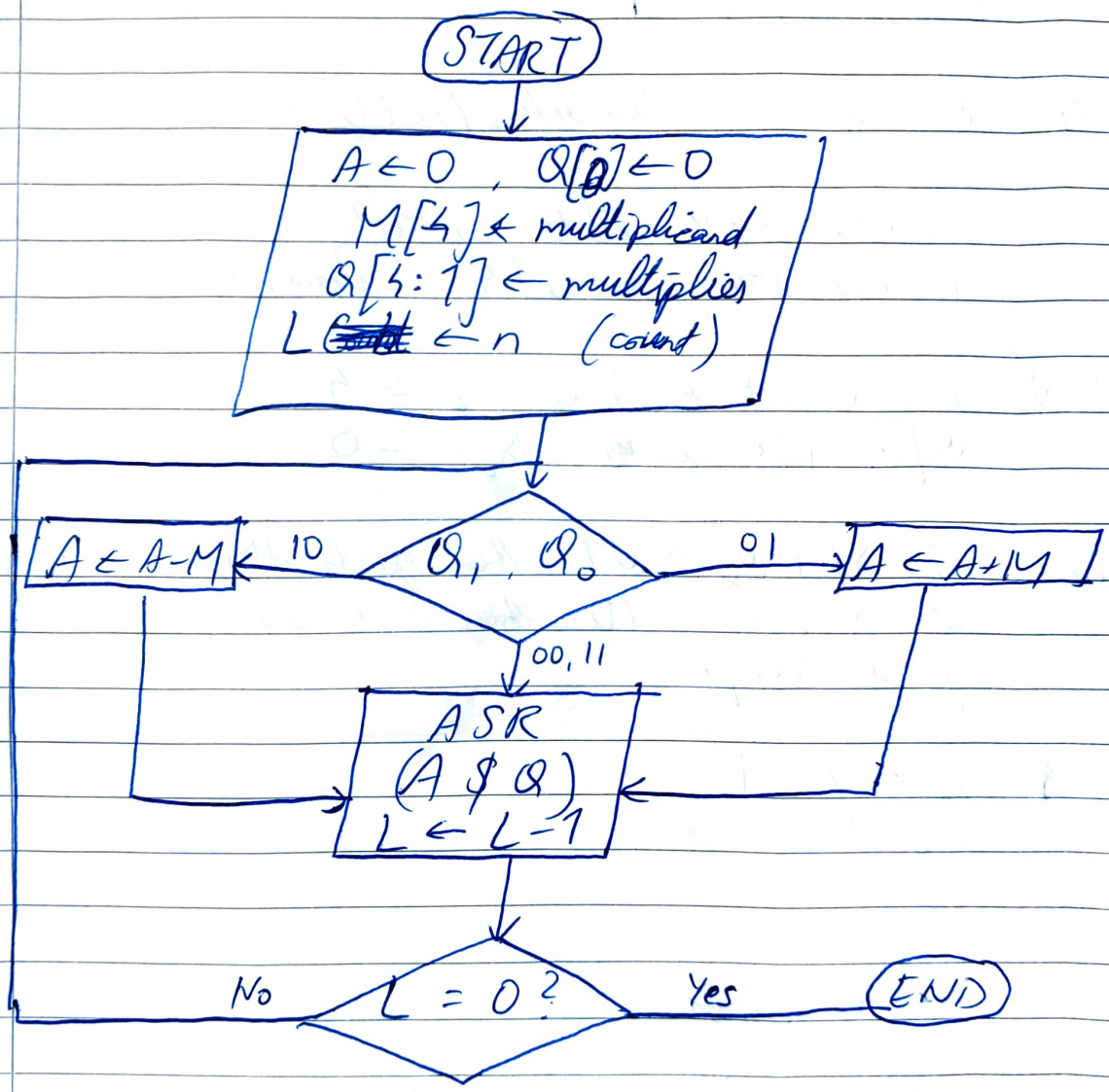
Design Simplicity $\Rightarrow$ $1 > 2 > 3$

__/__/__

☆ Hardwired Control Design

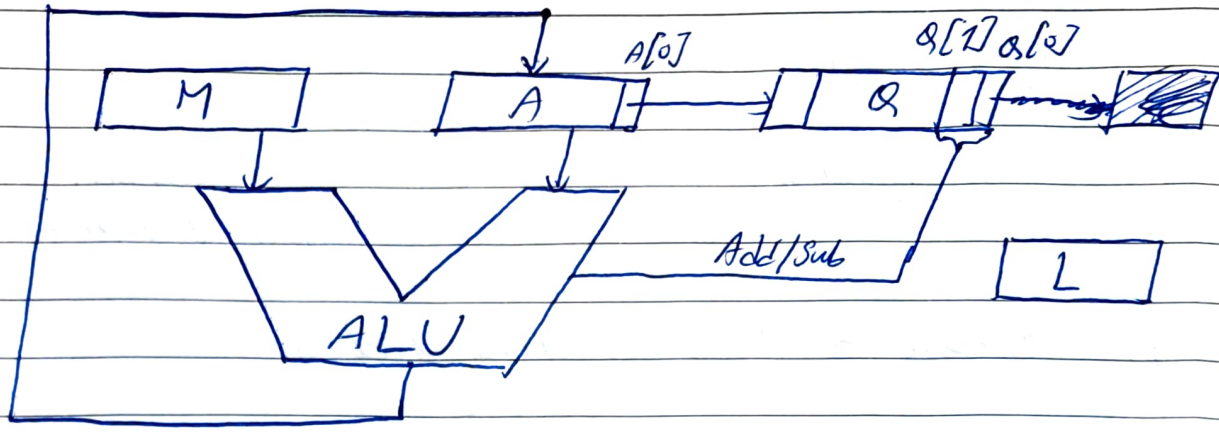
Step 1: Task Definition : Design a Booth's multiplier to multiply two 4-bit 2's complement numbers.

Step 2: Propose a trial processing section.



Q[1]	Q[0]	Action
0	0	None (ASR)
0	1	Add M
1	0	Subtract M
1	1	None (ASR)

__/__/____



Step 3 Register Transfer Description (RTD) of algorithm.

Declare registers A[4], M[4], Q[5], L[3];
Declare buses inbus[4], outbus[4];

Start: A ← 0 M ← inbus L ← 4;
Q[4:1] ← inbus, Q[0] ← 0;

Loop: if Q[1:0] = 01, then go to ADD;
if Q[1:0] = 10, then go to SUB;
go to Rshift

ADD: A ← A+M
Go to Rshift;

SUB: A ← A-M;
~~Go to Rshift;~~

Rshift: ASR (AQ), L ← L-1;
if L > 0, then Go to Loop
outbus = A;
outbus = Q[4:1];

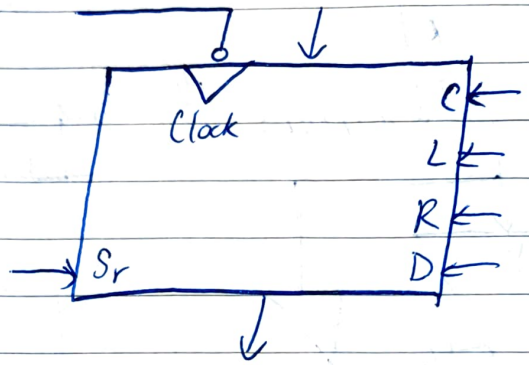
Halt: Go to Halt

__/__/____

Step4: Validate (using example)

Step5: Describe the basic characteristics of the hardware elements to be used in the processing section.

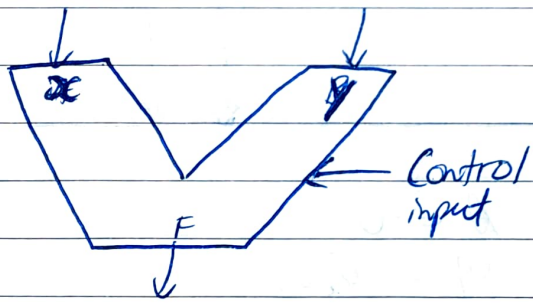
(a) General-purpose storage registers



C	L	R	D	Action
↑	0	0	0	Clear
0	↑	0	0	Load External Data
0	0	↑	0	Right shift
0	0	0	↑	Decrement (-1)
0	0	0	0	No change

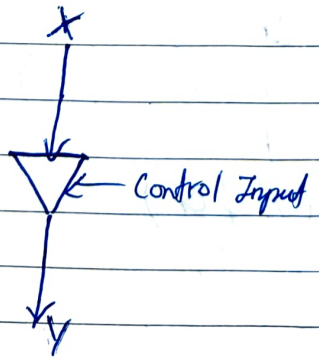
S_r → serial input for right shift.

(b) 4-bit adder/subtracter.



Control Input	F
1	x+y
0	x-y

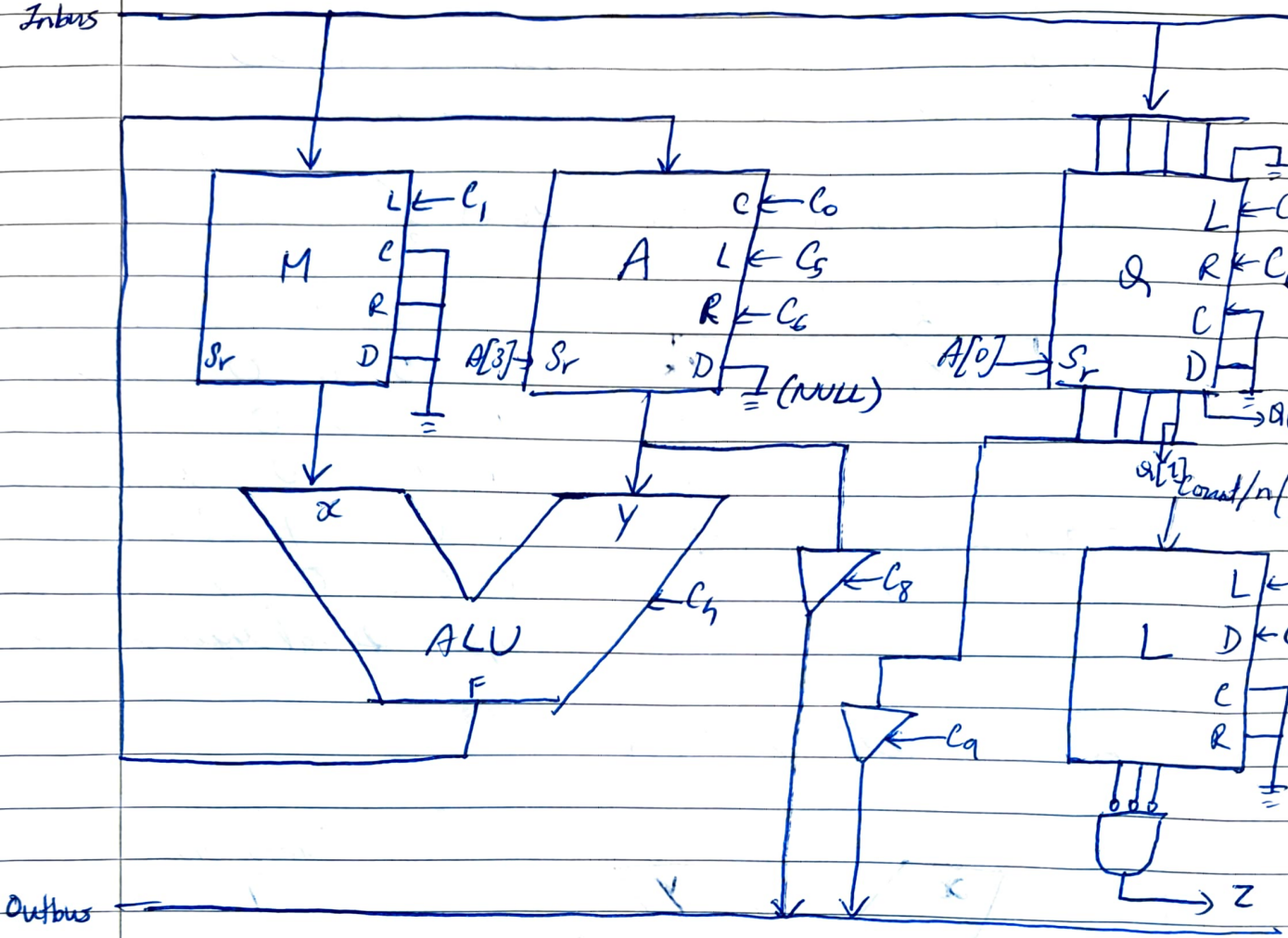
(c) Tristate buffers (used to control the data transfers to the outbus).



Control Input	y
1	x
0	High Z (no output)

__/__/____

Step 6: Complete design of processing section with required control points



Here C_0 : $A \leftarrow 0$

C_1 : $M \leftarrow$ inbus

C_2 : $L \leftarrow 4$

C_3 : $Q[4:1] \leftarrow$ inbus

$Q[0] \leftarrow 0$

C_4 : $F = x + y$ (1)

$F = x - y$ (0)

C_5 : $A \leftarrow F$

C_6 : ASR (A $ Q)

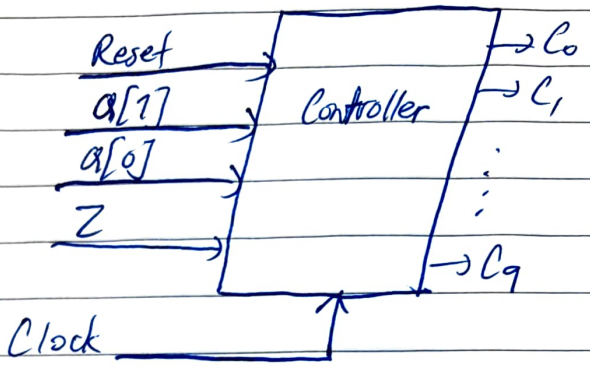
C_7 : $L \leftarrow L - 1$;

C_8 : Outbus = A

C_9 : Outbus = $Q[4:1]$

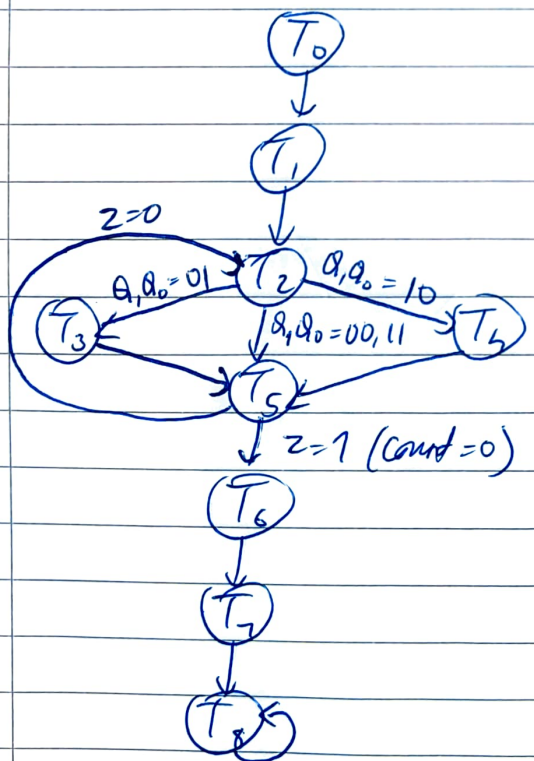
__/__/____

Step 7: Block diagram of controller.



- RESET input is an asynchronous input used to reset the controller, to start new computation. Clock input is used to synchronize controllers action.

Step 8: State diagram of controller.



State	Op	Control
T_0	$A \leftarrow 0$	$C_0, C_1,$
	$L \leftarrow 4$	C_2
	$M \rightarrow$ inbus	
T_1	$Q[4:1] \leftarrow$ inbus	C_3
	$Q[0] \leftarrow 0$	
T_2	—	—
T_3	$A \leftarrow A+M$	C_4, C_5
T_4	$A \leftarrow A-M$	C_5 ($C_4=0$)
T_5	ASR(A$Q)	C_6, C_7
	$L \leftarrow L-1$	
T_6	Outbus = A	C_8
T_7	Outbus = Q[4:1]	C_9
T_8	—	—

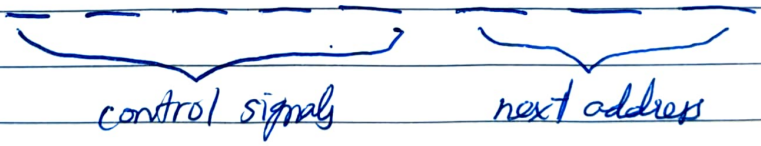
__/__/____

Microprogrammed Control Unit

- Control Word (CW): All control signals that can be simultaneously activated are grouped to form a CW.
- Each control word contains signals to activate one/more μprograms (A ← 0, B out bus ← A, etc.) and are stored in control memory (CM)
- After control words are fetched from CM, individual control fields are routed to various functional units to achieve desired tasks.
- All μinstructions have two fields; (a) Control field, to indicate which control signals to be activated and (b) Next address field to specify address of next μinstruction.

— Wilke's design

- Decoder and diode utilize an 8 × 8 ROM, each holding an 8-bit control word.



- Contents of ROM are retrieved by specifying it's address in CMAR (Control Memory Address Register)
- It is possible to load CMAR with externally specified new starting address. The external condition sets/resets the condition flipflop.

Ex:- If current ROM address = 5
External condition = 1, 5 → 1 (reset)
= 0, 5 → 6 (set)

- Length of μinstruction depends on: (a) degree of parallelism (b) control field organization (c) method of specifying next address.
- Sequential execution is handled by (+) μprogram counter.
- Branching (conditional / unconditional) handled by having fields in μinstruction that specify next address.

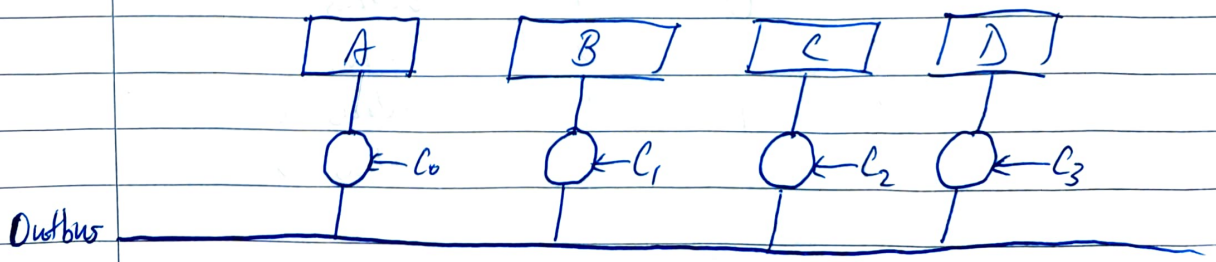
Microprogram

Horizontal
(Long μinstructions,
High degree of parallelism,
Less encoding)

Vertical
(Short μinstructions,
limited scoped parallelism,
More decoding)

– Encoding & Decoding

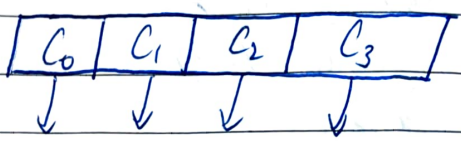
- Consider the simple registers transfer to outbus.



__/__/____

- Here C_0 : Outbus = A
  C_1 : Outbus = B
  C_2 : Outbus = C
  C_3 : Outbus = D

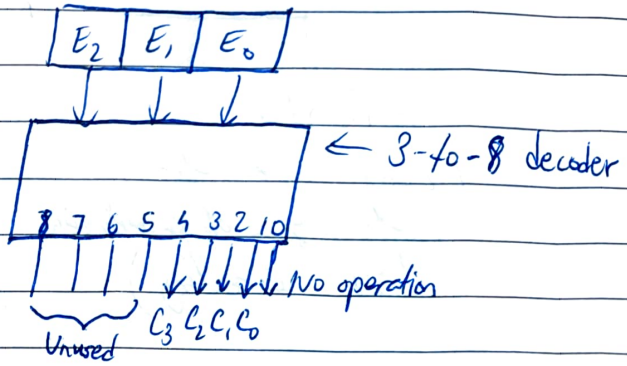
- Unencoded :



C_0	C_1	C_2	C_3	
1	0	0	0	Outbus = A
0	1	0	0	Outbus = B
0	0	1	0	Outbus = C
0	0	0	1	Outbus = D
0	0	0	0	No operation.

- Encoded :

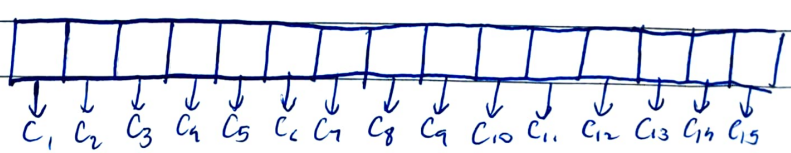
(shorter control field, short μinstructions)



E_2	E_1	E_0	
0	0	0	No operation
0	0	1	Outbus = A
0	1	0	Outbus = B
0	1	1	Outbus = C
1	0	0	Outbus = D

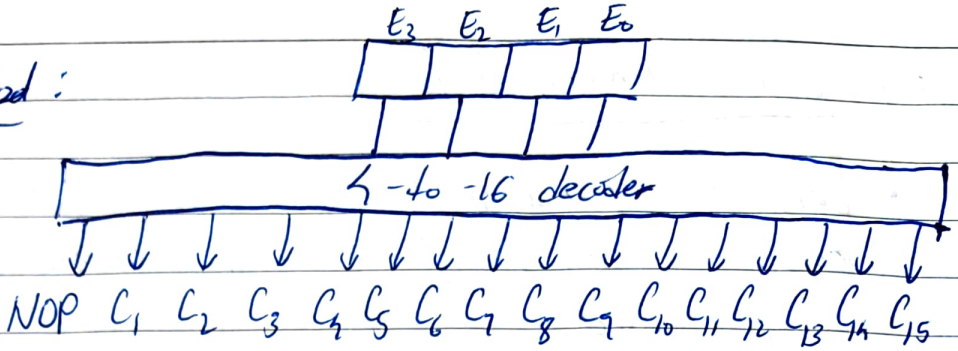
- For 15 control lines :

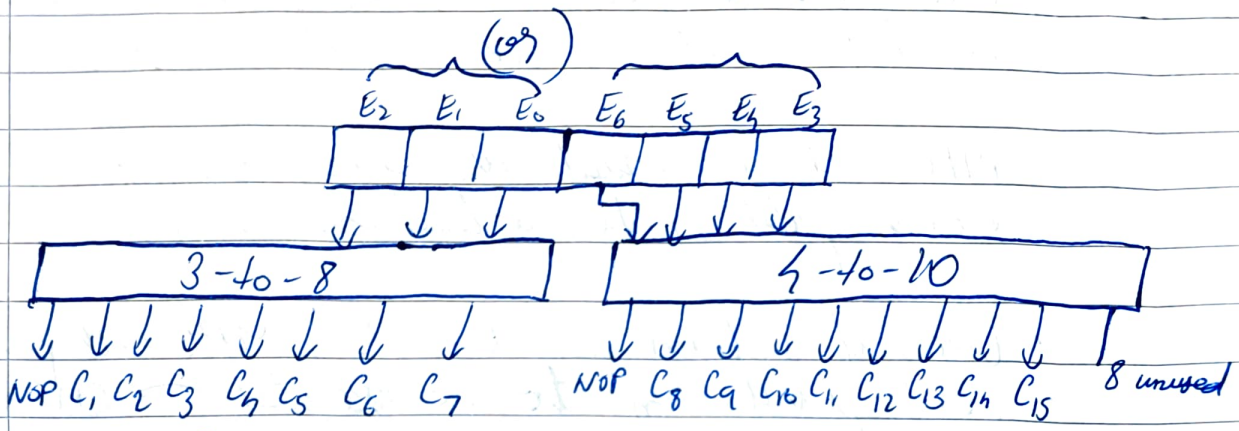
- Unencoded :



__/__/__

- Encoded:

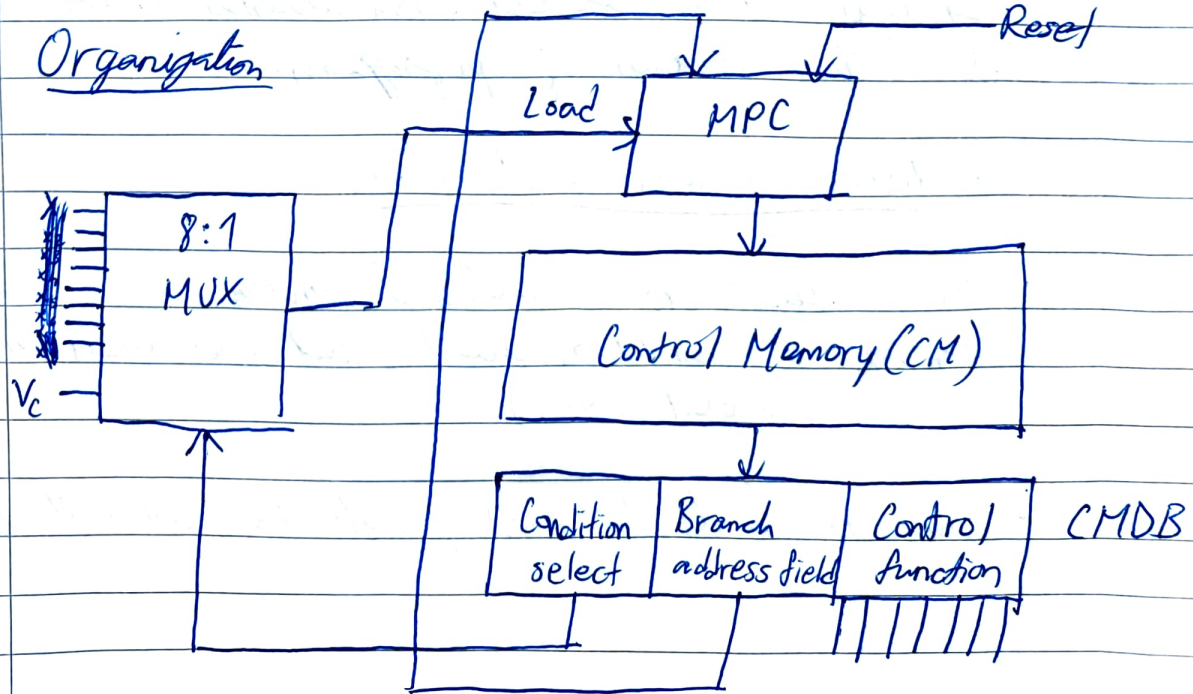




- Here, two signals of different groups can be activated in parallel. $C_1 \# C_8$ ✓
  $C_1 // C_2$ ✗

- The next-address field from microinstruction can be eliminated by using Micro Program Counter (MPC)

- Organization



__/__/____

(a) Control Memory Buffer Register (CMBR)

- Latch that functions similar to MBR of main memory, acts as buffer for μinstructions retrieved from CM.
- Control ~~function~~ select field selects the external conditions to be tested (acts as select input to MUX)
- MPC will be loaded with address specified in Branch Address field (when external condition = 1)
- Else, if external condition = 0, MPC will point to next μinstruction to be executed. → CMBR.
- Control function field holds control information in encoded form.

(b) Micro Program Counter

- Holds address of next μinstruction to be executed. Initially it is loaded from external source to point to starting address of μprogram.

(c) External Condition Select MUX

- MUX selects one of external conditions according to contents of condition select field. and loads in MPC. (in encoded form)
- Encoding leads to short μinstruction, less cost, small CM.

- Suppose for six external conditions,

Condition select	Action taken
000	No branching (Output(MUX)=0)
001	Branch if X1=1
010	Branch if X2=1
011	Branch if X3=1
100	Branch if X4=1
101	Branch if X5=1
110	Branch if X6=1
111	Branch always (unconditional)

- Here if 000 ⇒ Output(MUX)=0, MPC(+)
  010 ⇒ Output(MUX) = value of X2
  111 ⇒ Output(MUX) = 1, Branch address

Ex:- Design Microprogrammed Control Unit for Booth's Multiplier.

CM Address		Control Word
0	START	A ← 0, M ← inbus, L ← 4
1		Q[4:1] ← inbus, Q[0] ← 0
2	LOOP:	if Q[1:0] = 01 then go to ADD
3		if Q[1:0] = 10 then go to SUB
4		GO to RSHIFT
5	ADD	A ← A+M
6		Go to RSHIFT
7	SUB	A ← A-M
8	RSHIFT	ASR(A$Q), L ← L-1;
9		If Z=0, then go to LOOP
10		Outbus = A
11		Outbus = Q[4:1];
12	HALT:	Go to HALT

__/__/____

- CM hold 13 words

Conditional Select	Action taken
0 0 0	No branching
0 0 1	Branch if Q[1] Q[0] = 01
0 1 0	Branch if Q[1] Q[0] = 10
0 1 1	Branch if Z = 0
1 0 0	Unconditional branch

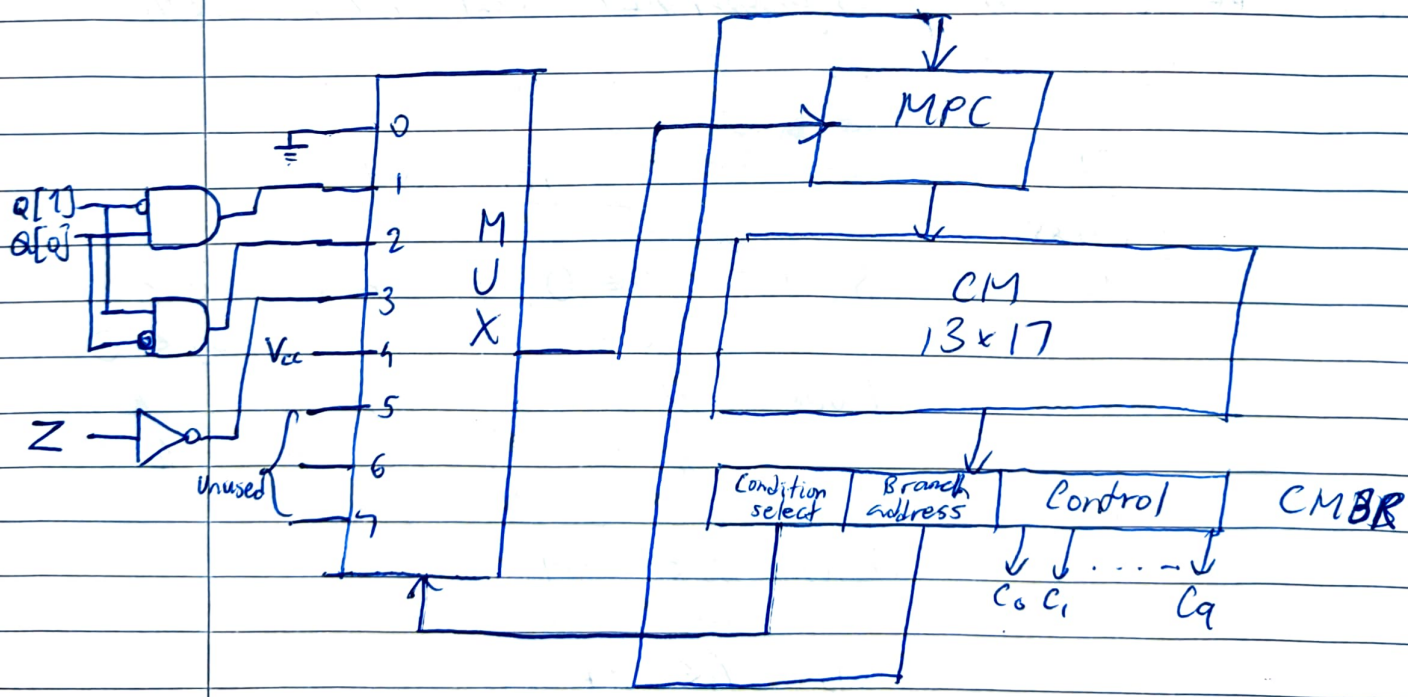
(Conditions)

Size of CM = Size of conditional select + Size of branch address (bits) + No. of function field. ($C_0 \rightarrow C_q$)

= 3 + 4 + 10

= 17 bits

Size of CMBR = 13 × 17 = 221 bits



__/__/____

ADDRESS		CONTROL WORD										
Dec.	Binary	Condition Select	Branch Address	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
0	0000	000	0000	1	1	1	0	0	0	0	0	0	0
1	0001	000	0000	0	0	0	1	0	0	0	0	0	0
2	0010	001	0101	0	0	0	0	0	0	0	0	0	0
3	0011	010	0111	0	0	0	0	0	0	0	0	0	0
4	0100	100	1000	0	0	0	0	0	0	0	0	0	0
5	0101	000	0000	0	0	0	0	1	1	0	0	0	0
6	0110	100	1000	0	0	0	0	0	0	0	0	0	0
7	0111	000	0000	0	0	0	0	1	0	0	0	0	0
8	1000	000	0000	0	0	0	0	0	0	1	1	0	0
9	1001	011	0010	0	0	0	0	0	0	0	0	0	0
10	1010	000	0000	0	0	0	0	0	0	0	1	0	0
11	1011	000	0000	0	0	0	0	0	0	0	0	0	1
12	1100	100	1100	0	0	0	0	0	0	0	0	0	0

Hardwired Control Unit	Microprogrammed Control Unit
• Generates control signals needed for processor using logic circuits.	• Generates control signals with the help of µ instructions stored in control memory.
• Faster.	• Slower
• Difficult to modify since it will require hardware redesign	• Easier to modify, changes can be made at µinstruction level.
• More expensive.	• Less expensive.
• Limited no. of instructions due to hardware constraints.	• Can support large number of instructions.
• Used in RISC.	• Used in CISC

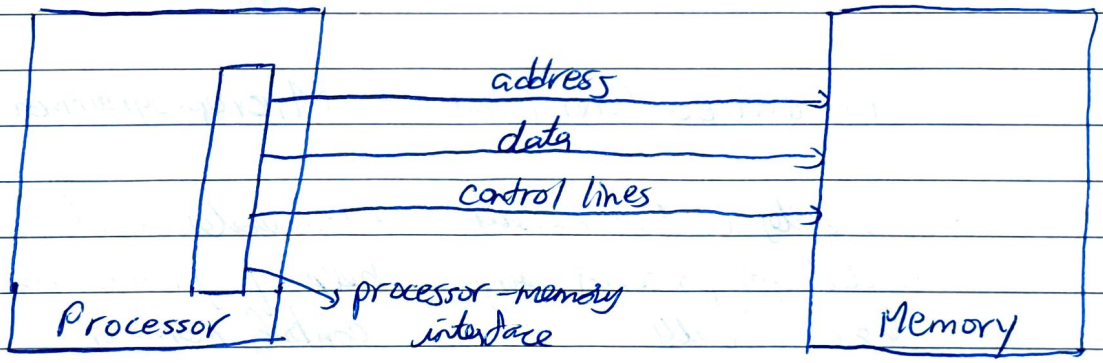
__/__/____

MEMORY SYSTEMS

- The maximum size of the memory that can be used in any computer is determined by addressing scheme.
- Number of locations represent size of address space of the computer.

Ex- If system uses 16-bit addresses, it can generate 2^{16} unique addresses, address space size is 65,536 memory locations or 64 KB (1KB = 1024 B)

- Memory is designed to store and retrieve data in word-length quantities.



- The processor uses address lines to specify memory locations involved in data-transfer operation, and uses data lines to transfer the data
- Control lines carry the command indicating Read or Write operation, and whether byte/word should be transferred.
- After receiving response from memory, the memory-function completed (MFC) signal is asserted and proceeds to the next step of execution.

__/__/____

- Memory access time is the time taken for system to retrieve data from memory after request is made, which includes time required to locate specific memory location and then transfer data to processor.

- Memory cycle time = (interval between start of one memory operation and start of next) + (memory access time) + (extra time to reset / prepare for next operation)

- Random-Access-Memory (RAM) is the type of computer memory that can be accessed randomly, any memory location can be read / written at same amount of time regardless of physical location in memory. Used to store data and instructions that are actively used / processed by CPU.

- Implemented using Semiconductor Integrated Circuits (IC's) which consist of multiple components such as transistors, capacitors, diodes all fabricated onto single piece of silicon.

- Since processor can process information faster than fetching from memory, one way to reduce memory access time is cache memory.

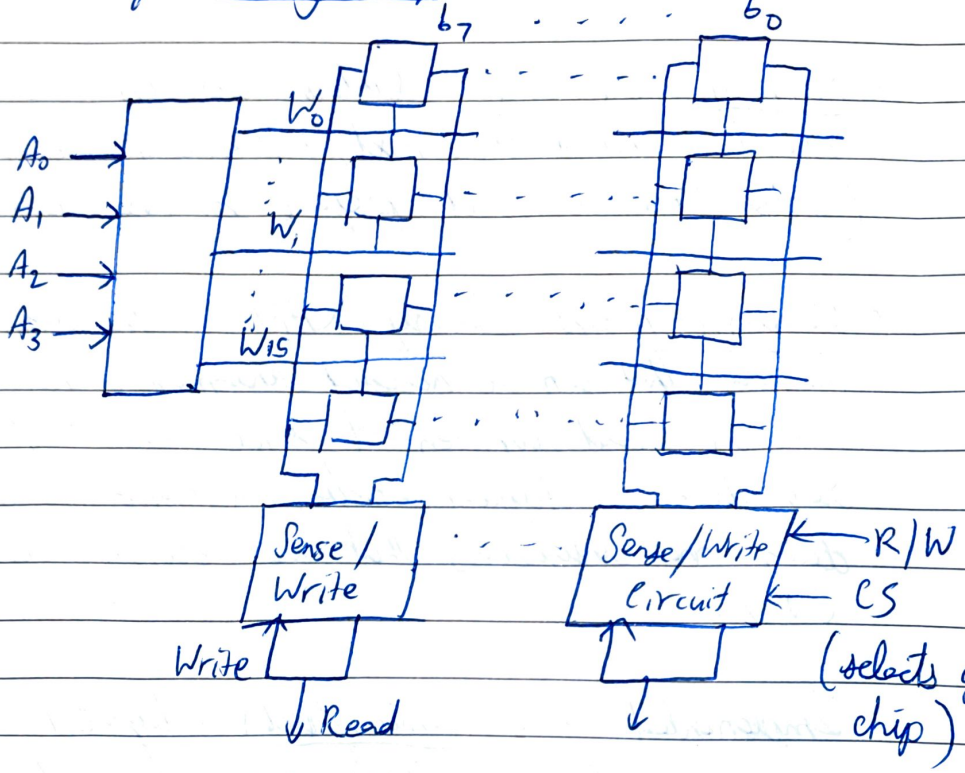
- Cache memory is small fast memory inserted between larger, slower main memory and processor, used to store currently active portions of program and data.

- Virtual memory is a technique in which active portions of program are stored in main memory and remainder stored in secondary storage device.

__/__/____

- Data is always transferred in blocks for high speed and efficient data transfer.

Internals of Memory Chip



- Memory cells are organized in the form of an array each cell capable of storing one bit information.
- Each row constitutes a memory word connected to a word line driven by address decoder on the chip.
- ~~Each~~ Cells in each column are connected to the Sense/Write circuit by two-bit lines.
- During Read operation, circuit senses/reads information stored in the cells selected by word lines and place it on output data lines.
- During Write operation, circuit ~~receives~~ input data and stores them in cells of selected word.

$2^{16} = 64\,KB = 64 \times 1024\ B$

$2^{32} = 4\,GB = 4 \times (1024)^3$

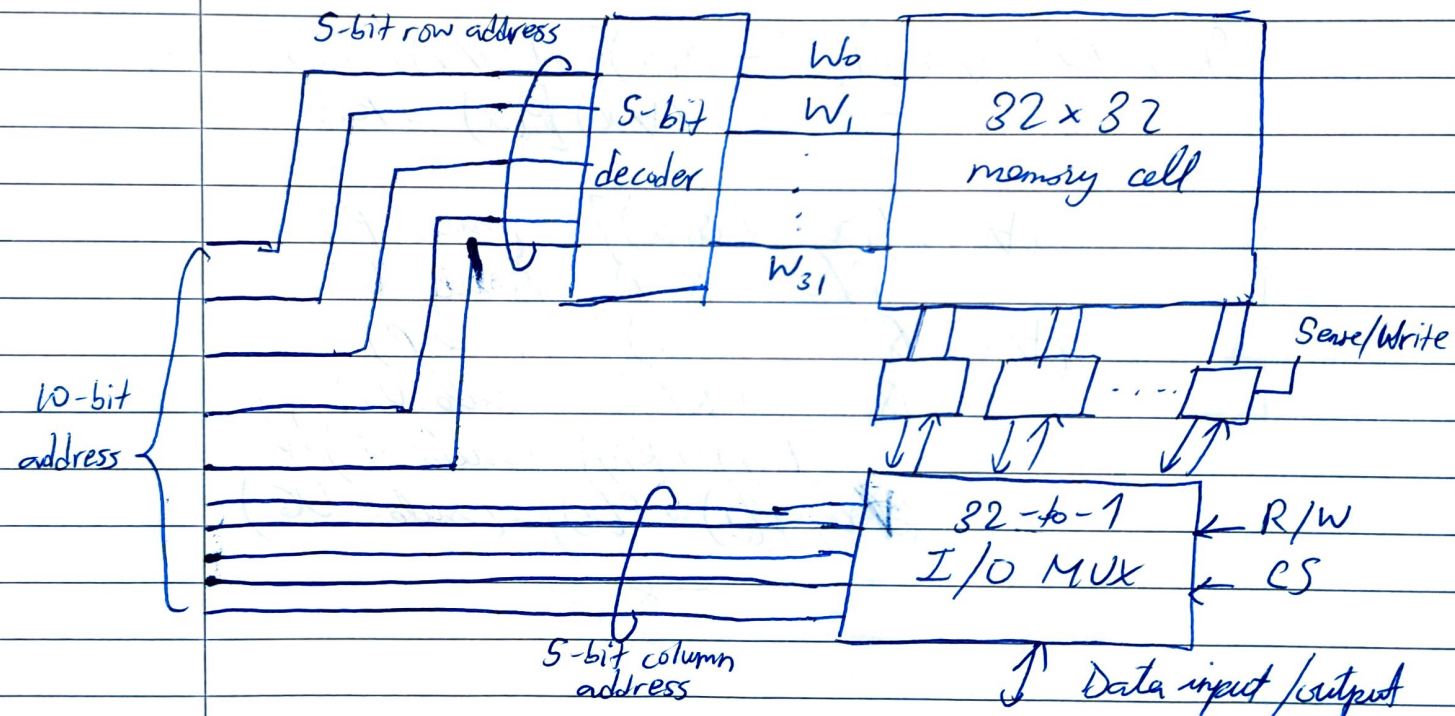
$MB \rightarrow (1024)^2$

__/__/____

COA (MODULE-5) (Contd.)

— Dynamic RAM

- If several memory words are organized in row, 10-bit address is divided into 5-bit row address which selects a row of 32 cells all accessed in parallel, and 5-bit column address, connecting only one of these cells to external data line via I/O MUX.



★ Large Memory Structures

— Static memory systems

- Consider a memory consisting of 2M words of 32-bits each implemented using 512K × 8 static memory chips.

- Each chip has a control input called Chip-Select.

__/__/____

- When input = 1, it enables chip to accept data from / to place data on data lines, while others are electrically disconnected from data lines.

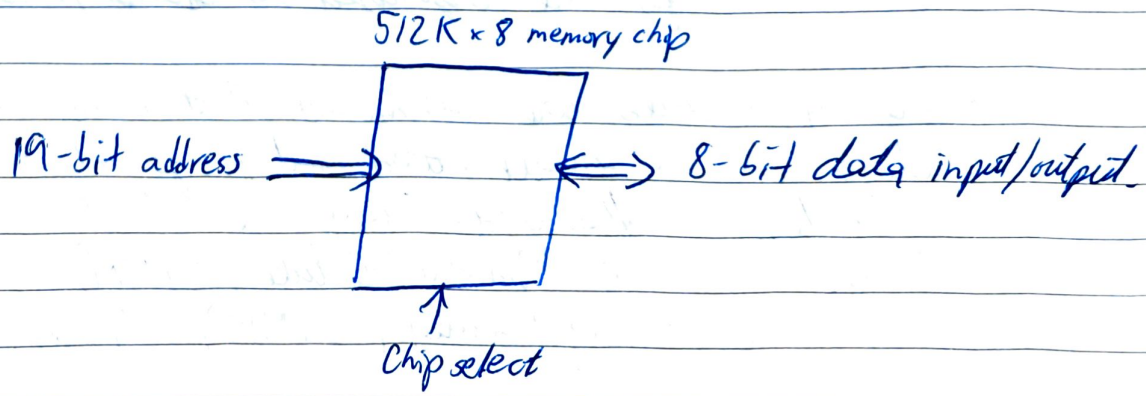
Step 1: No. of smaller chip to meet smaller size } = (row × column)
= Total required size / size of smaller chips
= $\frac{2M \times 32}{512K \times 8} = \frac{2 \times (1024)^2 \times 2^5}{512 \times 1024 \times 2^3}$
= $2^4 = 16$ chips

Step 2: No. of small chip to connect in // (no. of columns) ⇒
In $2M \times 32$, 32 bits required and in $512K \times 8$, 8 bits of data required, ⇒ $\frac{32}{8} = 4$ columns

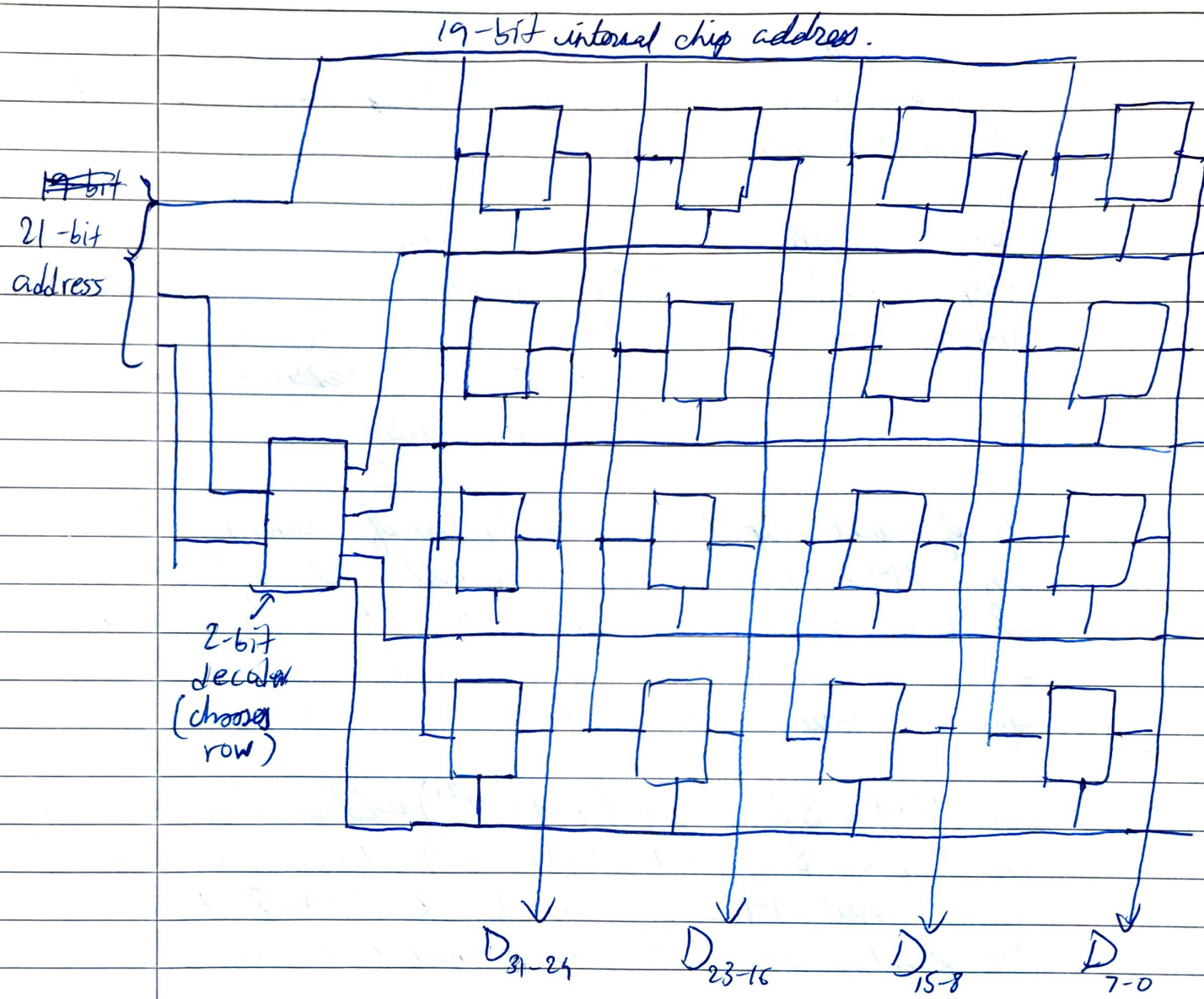
Step 3: No. of rows = $\frac{16}{4} = 4$

Step 4: In $2M \times 32$, 21 ($2M = 2^{21}$) address lines required.
In $512K \times 8$, 19 ($512K = 2^{19}$) address lines required.
Hence, first 19 lines connected to $512K \times 8$ chips
For selecting row, 2 lines connected to decoder

- From the selected row, 4 chips of $512K \times 8$ will read/write 8-bits of data each, making the size of 32 bits.



__/__/____



— Dynamic Memory Systems

- To reduce no. of memory chips in a given computer, memory chips are organized to read/write in parallel.
- Memory modules are assemblies that house many memory chips on a small board that plugs into a socket on computer's motherboard. They are of two types: Single In-Line Memory Module (SIMM) or Dual In-line Memory Modules (DIMM) depending on configuration.

__/__/__

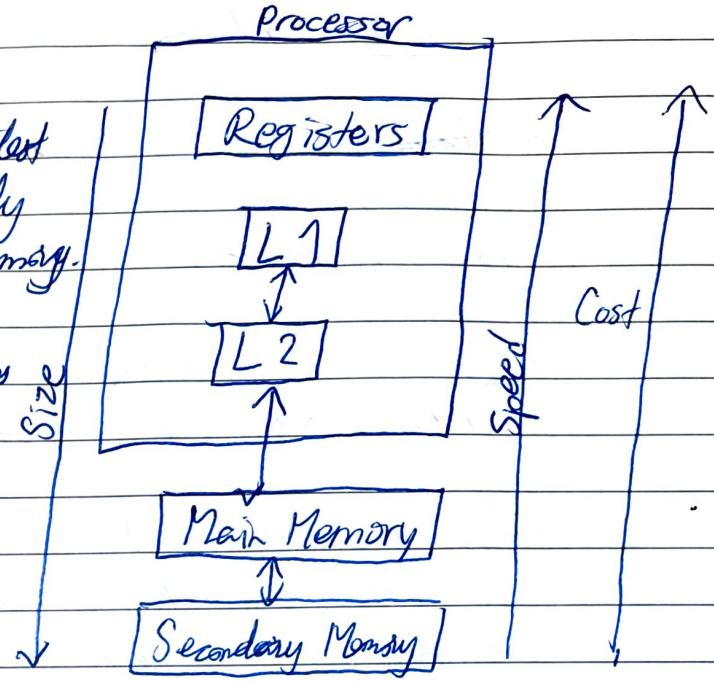
Memory Controller

- Since a typical processor issues all bits of an address at the same time, a MUX is required, with the help of memory control circuit (MCC)
- It accepts complete address and R/W signal from processor under control of Request signal.
- It forwards the R/W signals and row and column portions of address and generates the Row Address Strobe (RAS) signal and Column Address Strobe (CAS) signal.
- Dynamic RAM requires periodic refreshing, since they consist of capacitors & transistors. If capacitor leaks over time, data will be lost.
- Refresh Overhead = $\frac{(\text{No. of refresh cycles}) \times (\text{Access time})}{(\text{Refresh time})}$ (s)

SRAM	DRAM
• Can store data under power	• Saves data for a few moments w/o power.
• Low storage capacity	• High storage capacity
• More expensive	• Less expensive
• Faster and low power consumption.	• Slower and low cost of production
• Used in cache memory	• Used in main memories

__/__/____

☆ Memory Hierarchy



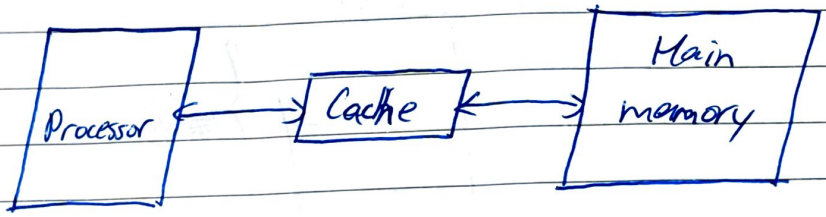
- Processor registers are fastest to access, but provide only small portion of required memory.
- Processor cache holds copies of instructions and data, implemented directly on processor chip.
  Ex: L1 cache, L2 cache.
- Main memory is the large memory implemented using dynamic memory components typically assembled in DIMMs (larger in size but slower)
- Secondary storage is inexpensive, slower compared to main memory but have the largest size.

☆ Cache Memories

- Cache is the small, fast memory interposed between processor and main memory.
- Locality of reference refers to phenomenon in which program tends to access same set of memory locations for a particular time period.
- Temporal locality suggests that whenever an item/data is first needed, it shall be brought to cache, since it is likely to be needed again soon. (LOOP)

__/__/____

- Special locality suggests that the adjacent addresses to the fetched item be brought to the cache memory as well.
- Cache block / line is a set of contiguous address locations of some size.



- Cache Hit

- Cache control circuitry determines whether requested word currently exists in cache or not, main memory is not involved. (during Read operation)
- For write operations, there are two main protocols:

(a) Write - Through (data simultaneously updated in both cache and main memory) (simple and reliable)

(b) Write - Back (only cache location updated, main memory location is updated later)

- Cache Miss

- Read operation for a word that is not in the cache constitutes a Read Miss.

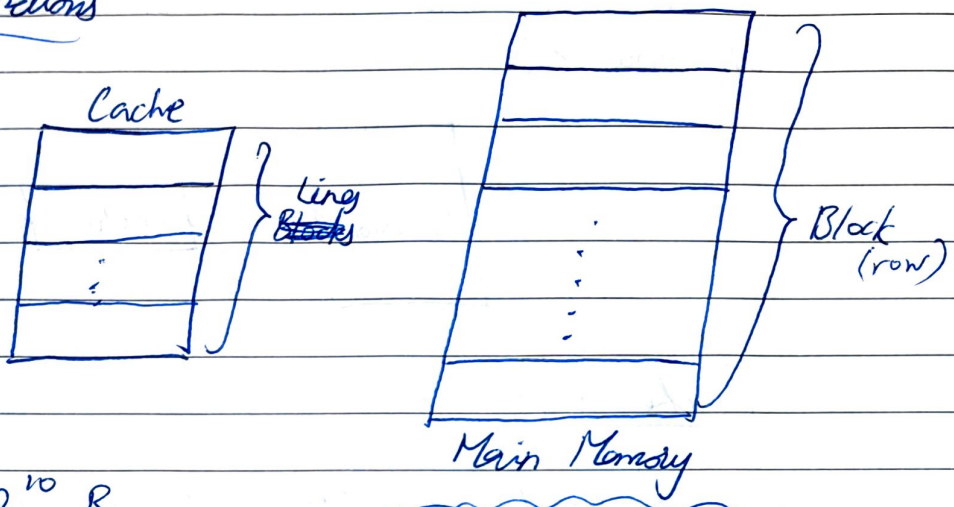
Main memory → Cache → Processor

- Load - Through (~~processor~~ main memory → processor) (reduces processor waiting time) (complex circuits)

__/__/____

- Write Miss uses write-through protocol, data is written directly into main memory.

— Mapping Functions



Note:
$1\,KB = 2^{10}\,B$
$1\,GB = 2^{30}\,B$
$1\,MB = 2^{20}\,B$

$\log_2(2^x) = \log_2(y)$
$x = \frac{\log y}{\log 2}$

① • Direct-Mapping

- Simplest way to determine cache locations in which to store memory blocks.
- Here, block j of main memory maps on block j%k (where k is no. of lines in cache) on cache.
- Since more than one memory block is mapped onto given cache block position, conflicts may arise. Can be resolved by new block overwriting current block.

Ex- Main Memory Size = $4\,GB = 2^2 \times 2^{30} = 2^{32}\,B$
Cache Size = $1\,MB = 2^{20}\,B$
Block Size = $4\,KB = 2^2 \times 2^{10} = 2^{12}\,B$

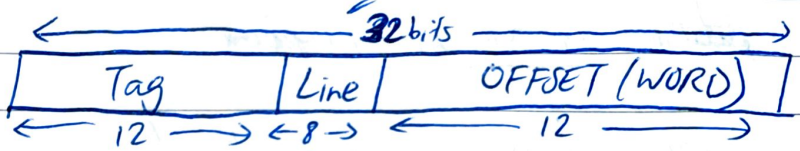
- To obtain the physical address split.

No. of physical address bits = $\log_2 2^{32} = 32$ bits
No. of blocks in MM = $\frac{2^{32}}{2^{12}} = 2^{20}$

No. of block bits = $\log_2 2^{20} = 20$ bits
No. of lines in Cache = $\frac{2^{20}}{2^{12}} = 2^8$

(Lines Size = Block Size)

No. of line bits = 8 bits
No. of tag bits = 32 - 20 - 8
= 12 bits



Tag Directory Size = Cache lines × Tag Bits
= $2^8 \times 12 = 3072$ bits

Note: • Tag bits identify the MM block residing in the Cache line.

Ex: • lower order bits select one of the words in the block.
• Cache line, locates the position of the word in the cache. address.

Ex: Cache : 128 blocks of 16 words each (total 2K words)
MM : 64 K (4K blocks of 16 words each)
Physical Address → 16 bits

Offset (words within block) ⇒ Since each block has 16 words each, $\log_2 16 = 4$ bits
Cache line ⇒ cache has 128 blocks, $\log_2(128) = 7$ bits
Tag ⇒ 16 - 7 - 4 = 5 bits

(5, 7, 4)

Cache has 128 blocks of 16 words each.

Differentiates → Tag | Block | Word
between (1 & 129) → 5 | log128=7 | log16=4

Ex: System uses 32-bit addresses.
MM Size ⇒ 1GB = 2^{30} bytes
Cache Size ⇒ 4KB = 2^{12} bytes
Block Size ⇒ 64B = 2^6 bytes

Offset ⇒ Each block has 64 bytes ⇒ $\log_2 64 = 6$ bits
Cache lines ⇒ Number = $\frac{2^{12}}{2^6} = 2^6 = 64$ blocks

$\log_2 64 = 6$ bits
Tag = 32 − 6 − 6 = 20 bits ∴ (20, 6, 6)

② ~~Address~~ Associative Mapping

- Here, a main memory block can be inserted into any cache block position, leading to more efficient use of space.
- For above example: | Tag (26) | Word (6) | (not easy to search, have to look at all tags)

③ Set Associative

- Blocks of cache are grouped into sets. Mapping allows block of MM to reside in any block of specific set.

Ex: If there are 64 ~~blocks~~ sets, then $\log_2 64 = 6$-bit set field will determine which set desired block is in.

- No. of blocks per set is a parameter which determines the technique used (direct / associative)
- Extreme conditions: One block / set → direct mapping
  All blocks in one set → associative

(MM block j) → (block j) % (No. of sets in cache)

For MM addressed by 16-bit address, $\frac{2^{16}}{2^4} = 2^{12}$ blocks.
(16 words/block)

$\frac{x}{16} = y$ (block in which x exists)
$y \% (\text{sets}) =$ set number in cache.

TAG	SET	WORD

Direct-Mapping	Associative-Mapping	Set-Associative
• Each block from MM has only one location in cache.	• A block from MM can be placed at any cache position.	• A block of data from MM can be place within a specific set.
TAG | LINE | WORD	TAG | WORD	TAG | SET | WORD
• Searching time is less	• Searching time is more.	• Searching time increase with increase in no. of sets.
• Simple & fast.	• Complex & slow	• Relatively complex.
• low cache-hit ratio	• Higher cache-hit ratio	• Medium cache-hit ratio

- Stale Data refers to the outdated / old data that no longer reflects the current state of system. It can be mitigated by regularly updating or refreshing cache data.

★ Replacement Algorithms

- When new block is to be brought into cache and all positions are full controller must decide which old block to overwrite (the one that has not been referenced for a long time) (which is called Least Recently Used (LRU) block)

PTO →

(i) Handling Cache Hits

- When processor request for block already present in cache,
  counter (referenced block) = 0
  counter (< ref) = (+1)

(ii) Handling Cache Misses (cache full)

- When processor requests for block not present in cache, block with highest counter value removed, replaced with new block, counter (ref) = 0, counter (remaining) (+1).

(iii) Handling Cache Misses (cache not full)

- If set is not full, new block added, counter (ref) = 0, counter (remaining) (+1).

Eg: CPU Reference → A → B → C → A → D → E → A → D → C → F

Miss Miss Miss Hit Miss Miss Hit Hit Hit Miss

A	A	A	(A)	A	A	(A)	A	A	A
	B	B	B	B	(E)	E	E	E	(F)
		C	C	C	C	C	C	(C)	C
				D	D	D	(D)	D	D
Counters	0	1	2	0	1	2	0	1	2	3
	1	0	1	2	3	0	1	2	3	0
	1	2	0	1	2	3	3	3	0	1
	1	2	3	3	0	1	2	0	1	2

Hit Ratio = $\frac{4}{10} = 0.4$

__/__/__

Ex: A byte-addressable computer has a small data cache capable of holding 8 blocks of 32-bit words each of the given HEX addresses, repeated 4-times. Assuming cache is empty initially, show contents of cache after each pass and compute hit rate.

200 — 0010000 000 00 → 0
204 — 0010000 001 00 → 1
208 — 0010000 010 00 → 2
20C — 0010000 011 00 → 3
2F4 — 0010000 101 00 → 5
2F0 — 0010000 100 00 → 4
218 — 0010000 110 00 → 6
21C — 0010000 111 00 → 7
24C — 0010000 011 00 → 3

- To address 8 blocks, $\log_2 8 = 3$ bits
- To address bytes within block ⇒ 32 bits = 4 bytes
  $\log_2 4 = 2$ bits

For HEX address, total bits = 12
Tag bits = 7 bits

TAG(7)	CACHE(3)	W(2)

__/__/____

Pass 1:

200 (Miss) (0)
204 (Miss) (1)
208 (Miss) (2)
20C (Miss) (3)
2F0 (Miss) (4)
2F4 (Miss) (5)
218 (Miss) (6)
21C (Miss) (7)
24C (Miss) (3)
(replace 20C)

Pass 2:

200 (Hit)
204 (Hit)
208 (Hit)
20C (Miss) (Replaced in (1))
2F0 (Hit)
2F4 (Hit)
218 (Hit)
21C (Hit)
24C (Miss) (Replaced now)

- Same happens in Pass 3 and 4 too. After 4 passes, total misses = 9 + 2 + 2 + 2 = 15

$$\text{Miss \%} = \frac{15}{12 \times 4} = \frac{15}{48} \times 100 = 31.25\%$$

48 ← total memory accesses

$$\text{Hit \%} = 69\%$$

★ Performance Considerations

- Common measure of success of a computer is price/performance ratio. Performance depends on how fast instructions can be brought into processor and how fast they can be executed.
- Hit rate is no. of hits (successful access to data in cache) as fraction of all attempted accesses.
- Miss rate is no. of misses as fractions of attempts.
- When miss occurs, extra time is needed to bring back data from slower to faster memory, during which processor is stalled

waiting for it, this is known as miss penalty.

Let h → hit rate, M → miss penalty,
C → time to access information in cache.
Average time (access) experience by processor,

$$t_{avg} = hC + (1-h)M$$

Ex: Access time to cache = τ
Access time to MM = 10τ
When cache miss occurs, block of 8 words transferred from MM → cache, 10τ seconds to transfer first word and τ/word for rest of 7 words, τ for initial access to cache, τ seconds for cache → processor.

$$M = \tau + 10\tau + 7\tau + \tau = 19\tau$$

Assuming 30% of instructions perform read/write operation, there are 130 memory accesses for every 100 instructions.
h(instructions) = 0.95, h(data) = 0.9.

$$\text{Improvement} = \frac{\text{Time w/o cache}}{\text{Time with cache}} = \frac{130 \times 10\tau}{100(0.95\tau + 0.05(19\tau)) + 30(0.9\tau + 0.1(19\tau))}$$

$$= 4.7$$

Compared to an ideal cache with 100% hit rate ($h = 1$)
(all memory references take one τ)

$$\frac{\text{Time for real cache}}{\text{Time for ideal cache}} = \frac{100(0.95\tau + 0.05(19\tau)) + 30(0.9\tau + 0.1(19\tau))}{130\tau}$$

$$= 2.1$$

↓
faster by 2x

- Hit rate can be improved by (a) making cache larger (high cost) (b) increasing cache block size keeping total cache size constant (spatial locality) (c) Larger blocks to load into cache in single miss, takes longer to transfer and increases miss penalty. (d) load-through approach.

- For multilevel cache in high performance processors such as L1 and L2 cache:

$$t_{avg} = h_1 C_1 + (1-h_1)(h_2 M_1 + (1-h_2) M_2)$$

where $h_1 \rightarrow$ hit rate in L1 cache
$h_2 \rightarrow$ hit rate in L2 cache
$C_1 \rightarrow$ time to access L1 cache
$M_1 \rightarrow$ miss penalty in L2 → L1
$M_2 \rightarrow$ miss penalty in MM → L2

- No. of misses in L2 cache $\Rightarrow (1-h_1)(1-h_2)$

Ex: Processor has 2 L1 caches & 1 L2 cache.

$C_1 = \tau$
$M_1 = 15\tau$, $M_2 = 100\tau$
$h_1 = 0.96$, $h_2 = 0.80$

(a) Misses in both L1 and L2 cache $= (1-0.96)(1-0.80)$
$= 0.008$

(b) $t_{avg} = 0.96\tau + 0.04\,(0.80\,(15\tau) + 0.20\,(100\tau)) = 2.24\tau$

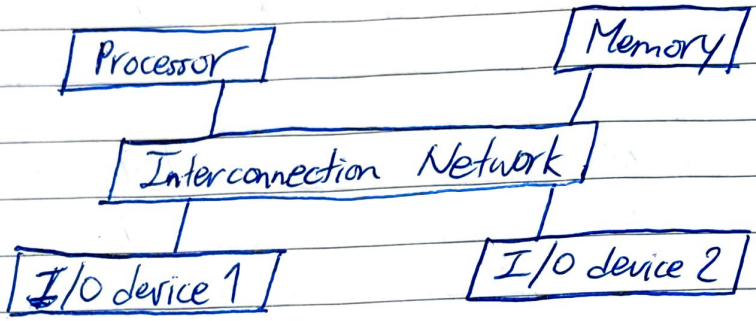
(c) $t_{ideal} = 0.96\tau + 0.04\,(15\tau) = 1.56\tau$

(d) $\frac{t_{avg}}{t_{ideal}} = 1.44$

COA

INPUT/OUTPUT ORGANIZATION

☆ Accessing I/O Devices



- Some addresses in address space of processor are assigned to I/O locations instead of main memory, implemented as bit-storage locations (flip-flops) in the form of a registers. ↳ I/O registers (Memory-mapped I/O)

```
Load  R2, DATAIN
Store R2, DATAOUT.
```

☆ I/O Device Interface

- I/O device is connected to interconnection network by using a device interface, which includes registers to serve as buffers for data transfer, registers to hold current state of device, and to store control information (behaviour) of device.

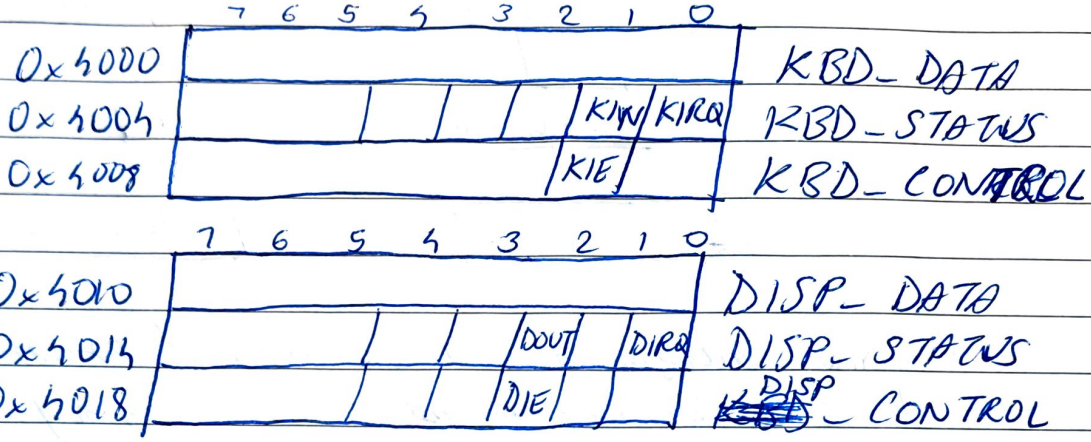
☆ Program-Controlled I/O

- It is a method in which program that performs all functions needed to realize desired action.

- Rate of data transfer from keyboard → process solely depends on typing speed of user.

__/__/____

- Rate of output transfer from computer to display is much higher than speed of processor in execution.
- One way to reduce transfer latency is signalling protocol



- A program is needed to perform the task of reading the characters produced by keyboard, storing these characters in memory and sending them to display.

INPUT:- When key is pressed, keyboard circuit places ASCII-encoded character into KBD_DATA register and sets KIN → 1. Processor detect KIN signal and transfers contents of KBD-DATA → processor register.

```
READWAIT    Read the KIN flag
            Branch to READWAIT if KIN=0
            Transfer data from KBD_DATA to R5
```

(or)

```
READWAIT    LoadByte  R4, KBD_STATUS
            And       R4, R4, #2      (test)
            Branch_if_[R4]=0  READWAIT
            LoadByte  R5, KBD_DATA
```

__/__/____

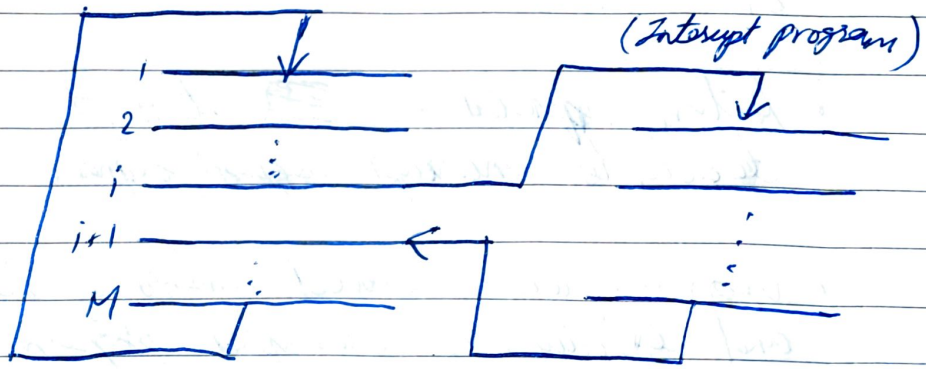
OUTPUT: When DOUT = 1 display is ready to recieve a character. Processor transfers ASCII-encoded character to DISP_DATA and DOUT → 0.

```
WRITEWAIT   Read the DOUT flag
            Branch to WRITEWAIT if DOUT=0
            Transfer data from R5 to DISP_DATA.
```

(or)

```
WRITEWAIT   LoadByte R4, DISP_STATUS
            And R4, R4, #4          (test)
            Branch_if_[R4]=0 WRITEWAIT
            StoreByte R5, DISP_DATA.
```

☆ Interrupts



- Sometime, hardware/software alerts the processor of a high-priority process needing immediate attention, causing an interruption in the current program being executed.
- As a result, after recieving the interrupt request processor stores the PC contents temporarily and loads the address of the Interrupt-Service-Routine (ISR), executes the required task and returns control to the interrupted task.
- When the processor loads the ISR it send a signal to the device to remove the interrupt request signal.

__/__/____

- Interrupt latency is the delay caused by saving registers and restoring interrupted process state from the time interrupt is recieved to start of ISR execution.

☆ Handling Interrupt Requests

- Device raises interrupt request.
- Processor interrupts program currently being executed and saves contents of PC and processor status (PS) registers.
- Interrupts are disabled by clearing IE (interrupt enable) bit in PS to 0.
- Action requested by ~~ISR~~ interrupt is performed by ISR, deactivates interrupt-request signal.
- Upon completion, saved contents of PC and PS are restored and execution of interrupted program is resumed.

☆ Handling Multiple Devices

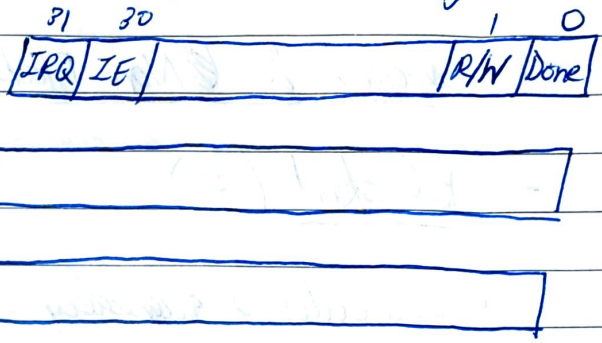
- Efficiently handling of interrupts from multiple devices involves identifying requesting device, determining the appropriate ISR, managing interrupt priorities and handling simultaneous requests.
- The processor can poll devices (Interrupt Request) to check their IRQ bits to identify requesting device, which is time consuming
- Vectored interrupts are a type of interrupt handling where device that generates interrupts provides the address

of the specific ISR directly into CPU. IVT is a table in memory that holds the address of all possible ISRs.

- Higher priority interrupts can interrupt low priority ISRs. (nested interrupt)

☆ Direct Memory Access (DMA)

- Allows data to be transferred directly between MM and I/O devices w/o intervention of CPU, reducing overhead.
- DMA controller:



- Starting Address register is where data transfer begins, Word Count is no. of words to transfer.
- Read/Write bit (1 → read, 0 → write)
  MM→I/O I/O→MM
- First, CPU initiates DMA transfer by providing starting address, word count and direction of transfer.
- DMA controller, then directly transfers data between I/O device and memory, and increments memory address and decrements word count.
- Upon completion, DMA controller updates status register of CPU. Transfer is done.

—x—