Unit III:
Control Unit Design

Control Unit Design

- There are two major types of control organization:
  1. Hardwired control
     - In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
     - It has the advantage that it can be optimized to produce a fast mode of operation.
     - It requires changes in the wiring among the various components if the design has to be modified or changed.
  2. Microprogrammed control
     - In the microprogrammed organization, the control information is stored in a control memory.
     - The control memory is programmed to initiate the required sequence of microoperations.
     - In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.

Control Unit Design

Block Diagram of Control Unit

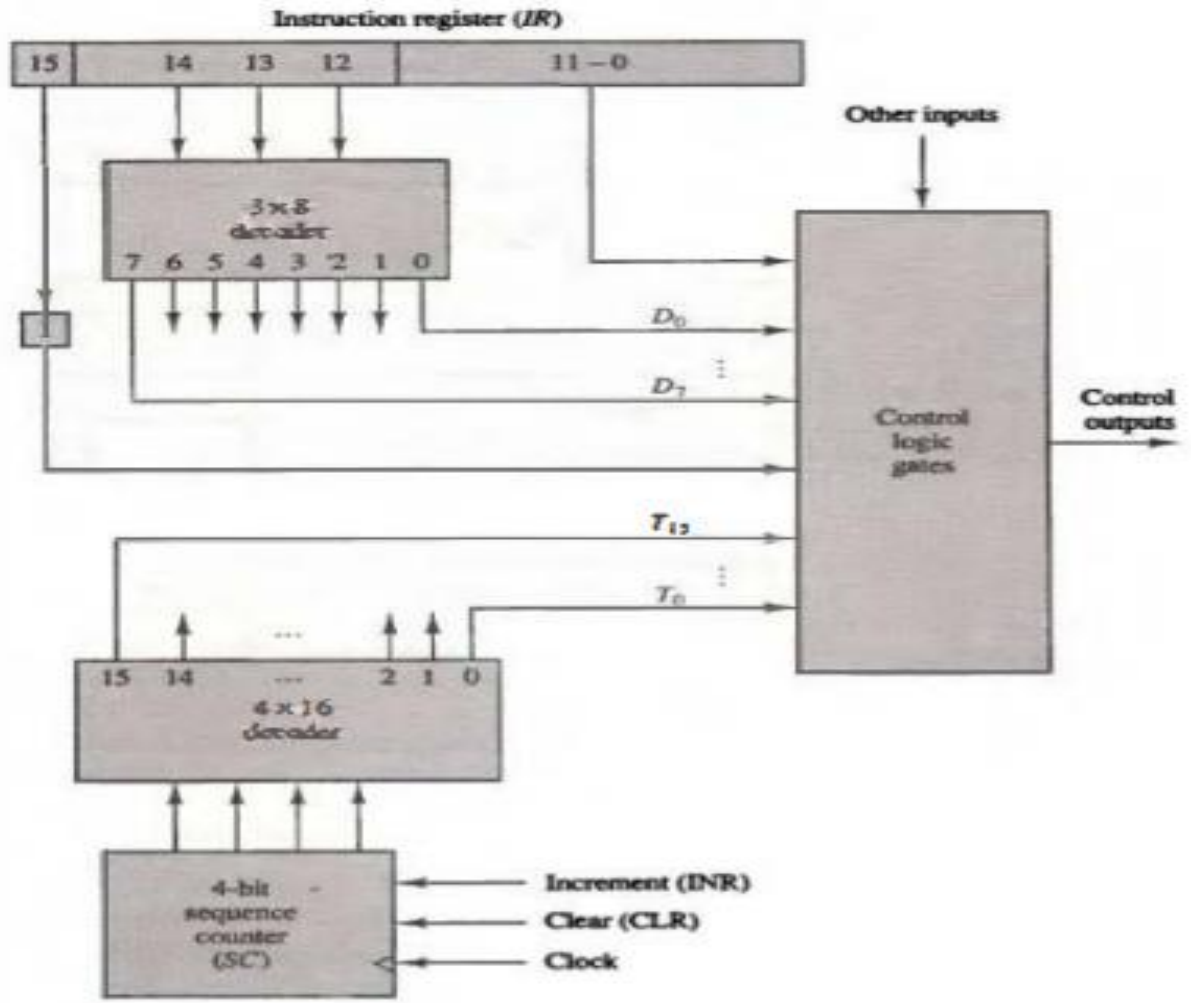


Figure: Control unit of Basic Computer

Control Unit Design

Instruction Cycle

- A program consists of a sequence of instructions.
- The program is executed in the computer by going through a cycle for each instruction.
- Each instruction cycle consists of the following phases:
  1. Fetch an instruction from memory.
  2. Decode the instruction.
  3. Read the effective address from memory if the instruction has an indirect address.
  4. Execute the instruction.
- Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Control Unit Design

Instruction Cycle

- **Fetch and decode**
- Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal T_o.
- After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2, and so on.
- The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

$$
\begin{aligned}
T_0&: \quad AR \leftarrow PC \\
T_1&: \quad IR \leftarrow M[AR], \quad PC \leftarrow PC + 1 \\
T_2&: \quad D_0, \ldots, D_7 \leftarrow \text{Decode } IR(12\text{–}14), \quad AR \leftarrow IR(0\text{–}11), \quad I \leftarrow IR(15)
\end{aligned}
$$

Control Unit Design

- **Instruction Fetch**
- To provide the data path for the transfer of PC to AR we apply timing signal T_0:
  1. Place the content of PC onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to 010.
  2. Transfer the content of the bus to AR by enabling the LD input of AR.
- To implement T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
  1. Enable the read input of memory.
  2. Place the content of memory onto the bus by making $S_2S_1S_0$ = 111.
  3. Transfer the content of the bus to IR by enabling the LD input of IR.
  4. Increment PC by enabling the INR input of PC.

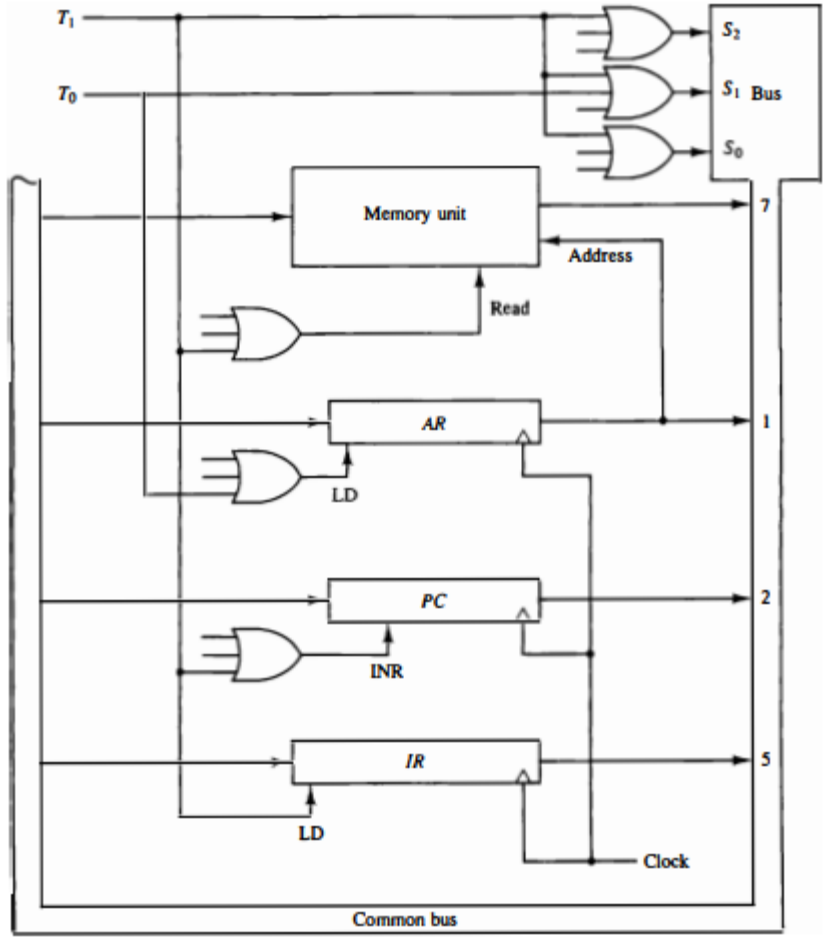


Figure: Register transfer for fetch phase.

Control Unit Design

- **Instruction Fetch, decode and execution.**

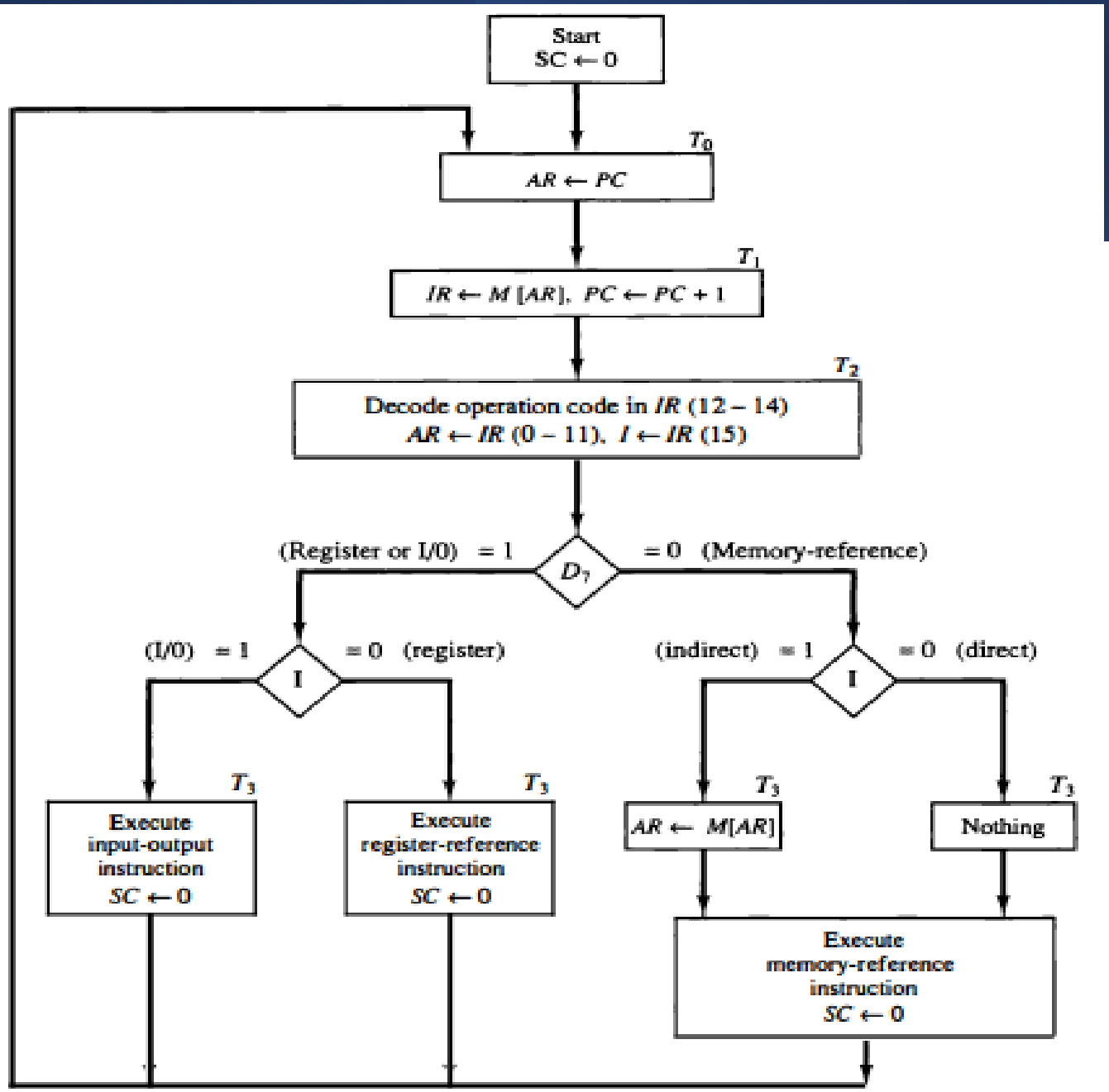
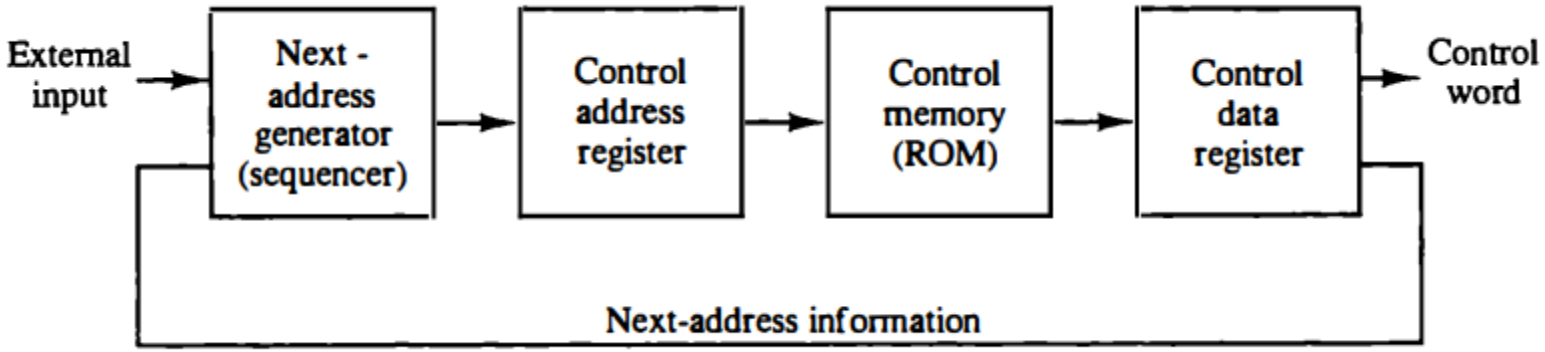


Figure: Flowchart for Instruction Cycle.

Control Unit Design

Microprogrammed Control Unit:



- The control memory is a ROM, within which all control information is permanently stored.
- Control memory address register specifies the address of the microinstruction.
- Control data register holds the microinstruction read from memory.

Control Unit Design

Microprogrammed Control Unit:

- A microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.
- The location of the next microinstruction mat be the one next in sequence, or it may be located somewhere else in control memory.
  - For this reason, it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction.
  - The next address may also be a function of external input conditions.
- While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

Control Unit Design

Microprogrammed Control Unit:

- The address of the next microinstruction can be specified in several ways, depending on the sequencer inputs. Typical functions of a microprogram sequencer are:
- Incrementing the control address register by one,
- Loading into the control address register an address from control memory, and
- transferring an external address or loading an initial address to start the control operations.

Control Unit Design

Microprogrammed Control Unit:

Advantage:

- The main advantage of the microprogrammed control is the fact that once the hardware configuration is established, there should be no need for further hardware or wiring changes.
  - If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory.
  - The hardware configuration should not be changed for different operations; the only thing that must be changed is the microprogram residing in control memory.

Control Unit Design

Microprogrammed Control Unit:

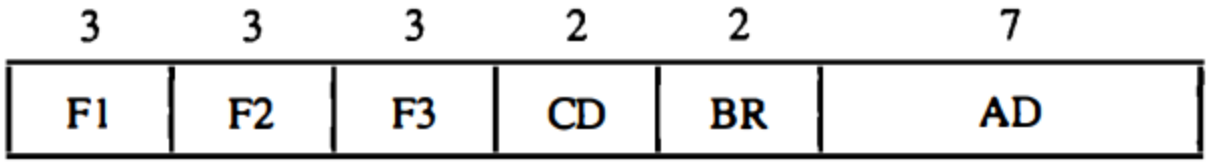
Disadvantage:

- Slower than a hardwired control unit.
  - Despite this, microprogramming is the dominant technique for implementing control units in pure CISC architectures, due to its ease of implementation.

Control Unit Design

Nano programming (or microprogramming)

- The process of generating the microcode for the control memory is known as microprogramming.
- Microinstruction Format:
- The 20 bits of the microinstruction are divided into four functional parts.



F1, F2, F3: Microoperation fields

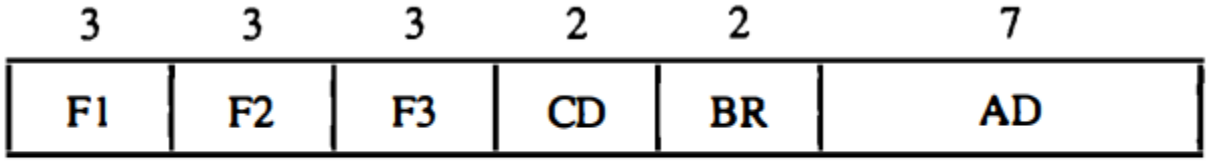
CD: Condition for branching

BR: Branch field

AD: Address field

Control Unit Design

Microprogramming



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

- F1, F2, and F3 specify microoperations for the computer.
- CD selects status bit conditions.
- BR specifies the type of branch to be used.
- AD contains a branch address. It is of 7 bits because control memory has 128 words.

Control Unit Design

Microprogramming

- **Symbols and Binary Code for Microinstruction Fields**

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Control Unit Design

Microprogramming

- **Symbols and Binary Code for Microinstruction Fields:**
- A microinstruction can specify two simultaneous microoperations. For example,

$$DR \leftarrow M[AR] \quad \text{with F2} = 100$$
$$\text{and} \quad PC \leftarrow PC + 1 \quad \text{with F3} = 101$$

- Microoperation fields: 000 100 101.
- Two or more conflicting microoperations cannot be specified simultaneously.
  - **For example**, 010 001 000 has no meaning because it specifies the operations to clear AC to 0 and subtract DR from AC at the same time.

Control Unit Design

Microprogramming

- **Symbols and Binary Code for Microinstruction Fields**

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	*DR*(15)	I	Indirect address bit
10	*AC*(15)	S	Sign bit of *AC*
11	*AC* = 0	Z	Zero value in *AC*

BR	Symbol	Function
00	JMP	*CAR* ← *AD* if condition = 1 *CAR* ← *CAR* + 1 if condition = 0
01	CALL	*CAR* ← *AD*, *SBR* ← *CAR* + 1 if condition = 1 *CAR* ← *CAR* + 1 if condition = 0
10	RET	*CAR* ← *SBR* (Return from subroutine)
11	MAP	*CAR*(2–5) ← *DR*(11–14), *CAR*(0,1,6) ← 0

Control Unit Design

Microprogramming

- All transfer-type **microoperations symbols** use five letters.
- The first two letters designate the source register, the third letter is always a T, and the last two letters designate the destination register.
  - **For example**, the microoperation that specifies the transfer AC <- DR (F1 = 100) has the symbol DRTAC, which stands for a transfer from DR to AC.

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

Control Unit Design

Microprogramming

- The **CD** (condition) **field** consists of two bits which are encoded to specify four status bit conditions.
  i. The first condition is always a 1, so that a reference to CD = 00 (or the symbol U) will always find the condition to be true.
     - When this condition is used in conjunction with the BR (branch) field, it provides an unconditional branch operation.
  ii. The indirect bit I is available from bit 15 of DR after an instruction is read from memory.
  iii. The sign bit of AC provides the next status bit.
  iv. The zero value, symbolized by Z, is a binary variable whose value is equal to 1 if all the bits in AC are equal to zero.
- Symbols U, I, S, and Z are used for the four status bits to write microprograms in symbolic form.

Control Unit Design

Microprogramming

- **BR field** consists of two bits and used in conjunction with the address field AD, to choose the address of the next microinstruction.
  i. When BR = 00, the control performs a jump (JMP) operation, and when BR = 01, it performs a call to subroutine (CALL) operation.
  ii. The two operations are identical except that a call microinstruction stores the return address in the subroutine register SBR.
  iii. The return from subroutine is accomplished with a BR field equal to 10.
  iv. The mapping from the operation code bits of the instruction to an address for CAR is accomplished when the BR field is equal to 11.

Control Unit Design

Microprogramming

- **Fetch Routine**
- The microinstructions needed for the fetch routine are:

$$AR \leftarrow PC$$

$$DR \leftarrow M[AR], \quad PC \leftarrow PC + 1$$

$$AR \leftarrow DR(0\text{–}10), \quad CAR(2\text{–}5) \leftarrow DR(11\text{–}14), \quad CAR(0,1,6) \leftarrow 0$$

Control Unit Design

Microprogramming

- **Fetch Routine**
- The address of the instruction is transferred from PC to AR.
- The instruction is read from memory into DR.
  - No instruction register is available, the instruction code remains in DR .
- The address part is transferred to AR and then control is transferred to one of 16 routines by mapping the operation code part of the instruction from DR into CAR.

Control Unit Design

Microprogramming

- **Fetch Routine**
- The symbolic microprogram for the fetch routine:

```
        ORG 64
FETCH:  PCTAR           U    JMP    NEXT
        READ, INCPC     U    JMP    NEXT
        DRTAR           U    MAP
```

- The translation of the symbolic microgram to binary produces the following microgram:

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

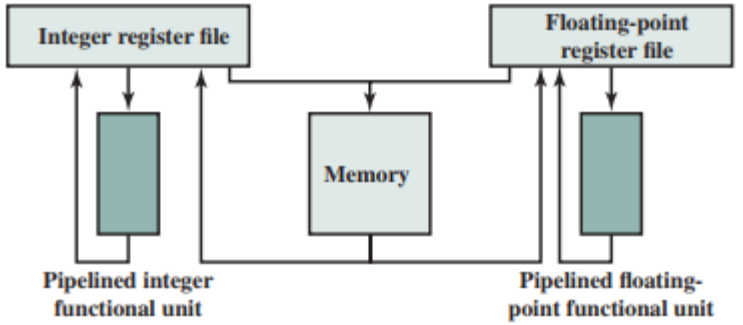
Control Unit Design

Superscalar Processing

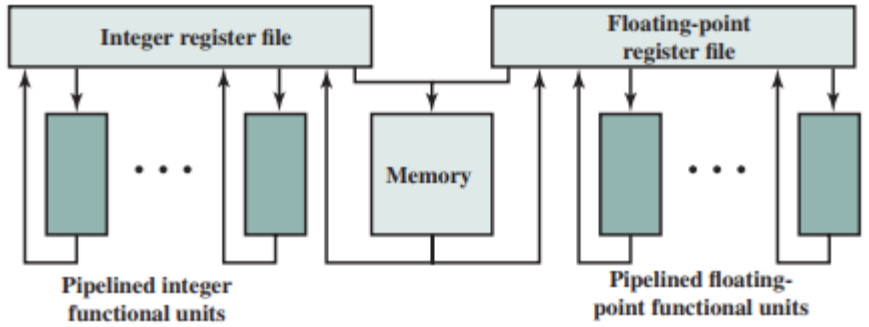
- A superscalar implementation of a processor architecture is one in which common instructions - integer and floating-point arithmetic, loads, stores, and conditional branches - can be initiated simultaneously and executed independently.

Control Unit Design

Scalar and Superscalar Organization



(a) Scalar organization



(b) Superscalar organization

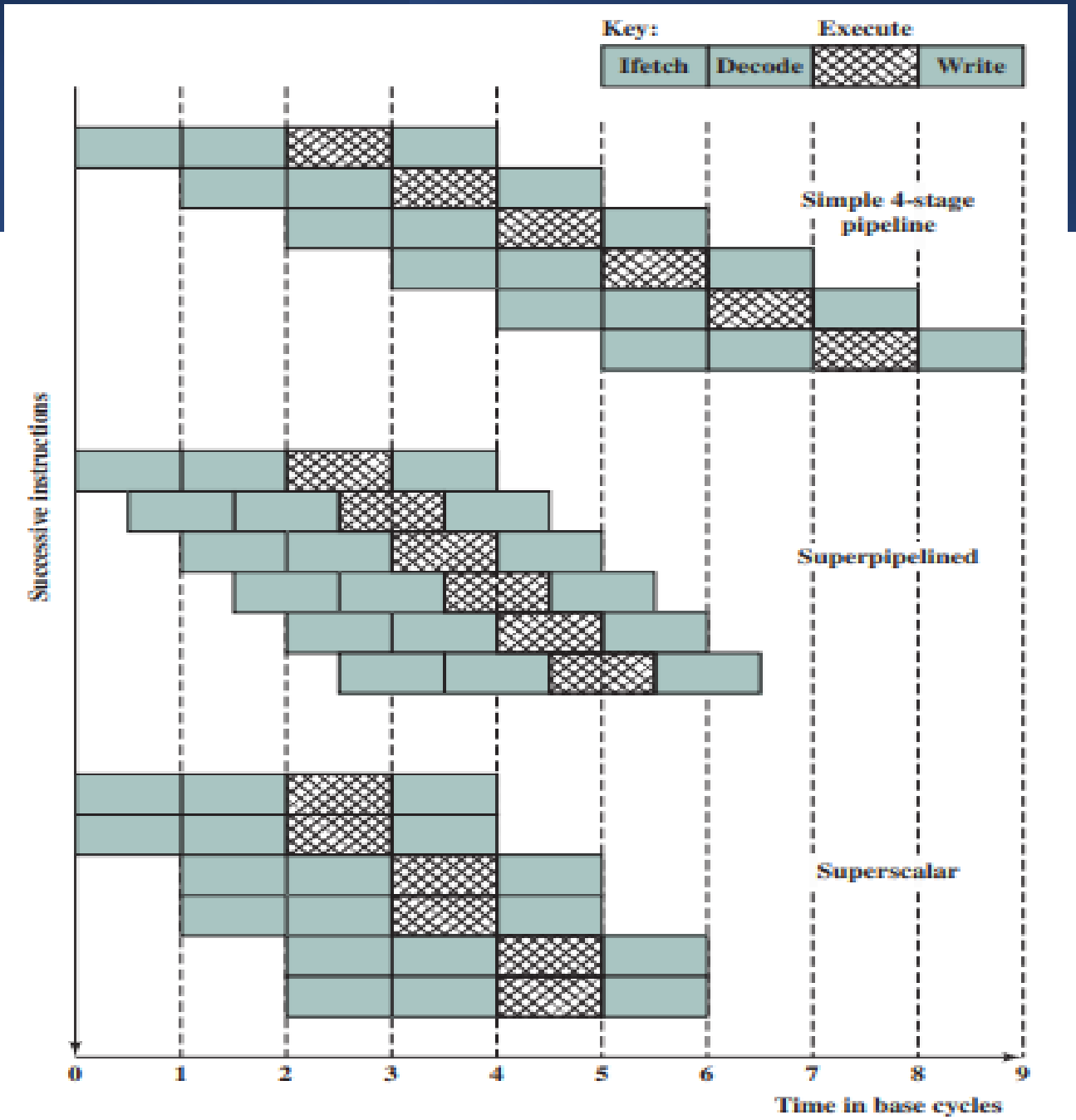
Control Unit Design

Scalar and Superscalar Organization

- In scalar organization, there is a single pipelined functional unit for integer operations and one for floating-point operations.
- In the superscalar organization, there are multiple functional units, each of which is implemented as a pipeline.
- Each individual functional unit provides a degree of parallelism by virtue of its pipelined structure.
- The use of multiple functional units enables the processor to execute streams of instructions in parallel.

Control Unit Design

Scalar, Superpipelined, and Superscalar Organization



Control Unit Design

Superscalar Organization

- Superscalar approach depends on the ability to execute multiple instructions in parallel.
- Five limitations to parallelism:
  i. True data dependency;
  ii. Procedural dependency;
  iii. Resource conflicts;
  iv. Output dependency
  v. Anti-dependency.

Control Unit Design

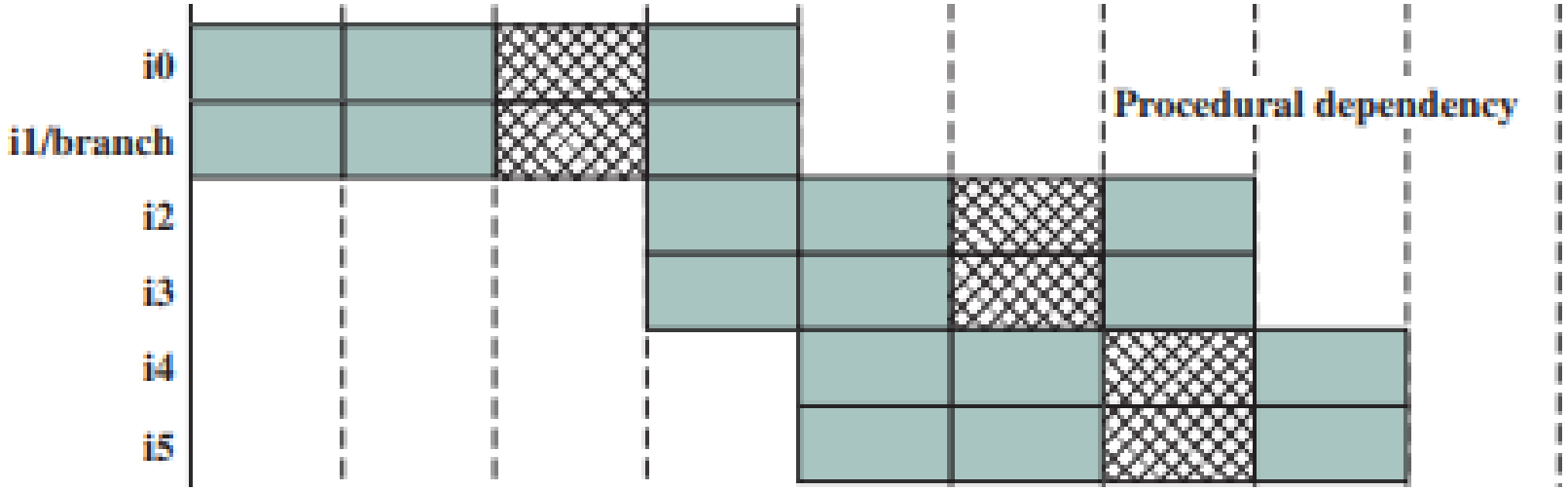
True data dependency

```
ADD EAX, ECX ;
             ;
             ;
MOV EBX, EAX ;
```

- Also called **flow dependency** or read after write (**RAW**) dependency.
- The second instruction can be fetched and decoded but cannot execute until the first instruction executes.

Control Unit Design

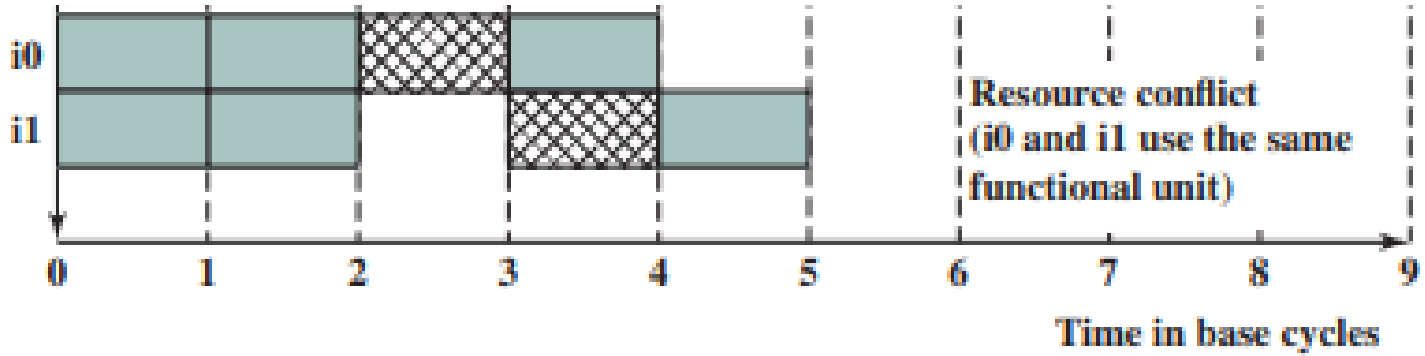
Procedural dependency



- The instructions following a branch (taken or not taken) have a **procedural dependency** on the branch and cannot be executed until the branch is executed.

Control Unit Design

Resource Conflict



- A resource conflict is a competition of two or more instructions for the same resource at the same time.
  - Examples of resources include memories, caches, buses, register-file ports, and functional units (e.g., ALU adder).
- Solution: Resource conflicts can be overcome by duplication of resources.

Control Unit Design

Output Dependency

```
I1: R3 ← R3 op R5
I2: R4 ← R3 + 1
I3: R3 ← R5 + 1
I4: R7 ← R3 op R4
```

- Also known as write after write (**WAW**) dependency.
- There is WAW dependency between I1 and I3, if I3 executes to completion prior to I1 then the wrong value of the contents of R3 will be fetched for the execution of I4.
- **Solution**: Issuing an instruction must be stalled if its result might later be overwritten by an older instruction that takes longer to complete; OR use register renaming.

Control Unit Design

Anti Dependency

```
I1: R3 ← R3 op R5
I2: R4 ← R3 + 1
I3: R3 ← R5 + 1
I4: R7 ← R3 op R4
```

- Also known as write after read (**WAR**) dependency.
- Instruction I3 cannot complete execution before instruction I2 begins execution and has fetched its operands.
  - Because I3 updates register R3, which is a source operand for I2.
- **Anti-dependency**: Instead of the first instruction producing a value that the second instruction uses; the second instruction destroys a value that the first instruction uses.

Control Unit Design

Register Renaming

```
I1: R3b ← R3a op R5a
I2: R4b ← R3b + 1
I3: R3c ← R5a + 1
I4: R7b ← R3c op R4b
```

- The creation of register R3c in instruction I3 avoids the WAR dependency on the second instruction and the WAW on the first instruction, and it does not interfere with the correct value being accessed by I4.
- Result: I3 can be issued immediately; without renaming, I3 cannot be issued until the first instruction is complete and the second instruction is issued.

Unit IV:
Memory Organization

Memory Organization

Semiconductor Memory

- It is a type of electronic data storage that uses semiconductor devices, typically integrated circuits (ICs), to store and retrieve digital information.

<table>
<tr><th>Memory Type</th><th>Category</th><th>Erasure</th><th>Write Mechanism</th><th>Volatility</th></tr>
<tr><td>Random-access memory (RAM)</td><td>Read-write memory</td><td>Electrically, byte-level</td><td>Electrically</td><td>Volatile</td></tr>
<tr><td>Read-only memory (ROM)</td><td rowspan="2">Read-only memory</td><td rowspan="2">Not possible</td><td>Masks</td><td rowspan="6">Nonvolatile</td></tr>
<tr><td>Programmable ROM (PROM)</td><td rowspan="5">Electrically</td></tr>
<tr><td>Erasable PROM (EPROM)</td><td rowspan="4">Read-mostly memory</td><td>UV light, chip-level</td></tr>
<tr><td>Electrically Erasable PROM (EEPROM)</td><td>Electrically, byte-level</td></tr>
<tr><td>Flash memory</td><td>Electrically, block-level</td></tr>
</table>

Figure: Semiconductor memory types.

Memory Organization

Semiconductor Memory

- **RAM**
- It is possible both to read data from the memory and to write new data into the memory easily and rapidly.
- Reading and writing are accomplished through electrical signals.
- Volatile – if the power is interrupted, then the data are lost.
- Types: DRAM and SRAM.

Memory Organization

Semiconductor Memory

- **RAM**
- DRAM is made with cells that store data as charge on capacitors.
- The presence or absence of charge in a capacitor is interpreted as a binary 1 or 0.
- Dynamic RAMs require periodic charge refreshing to maintain data storage because capacitors have a natural tendency to discharge.
- DRAM is more dense and less expensive than SRAM.
- DRAM is used for main memory.

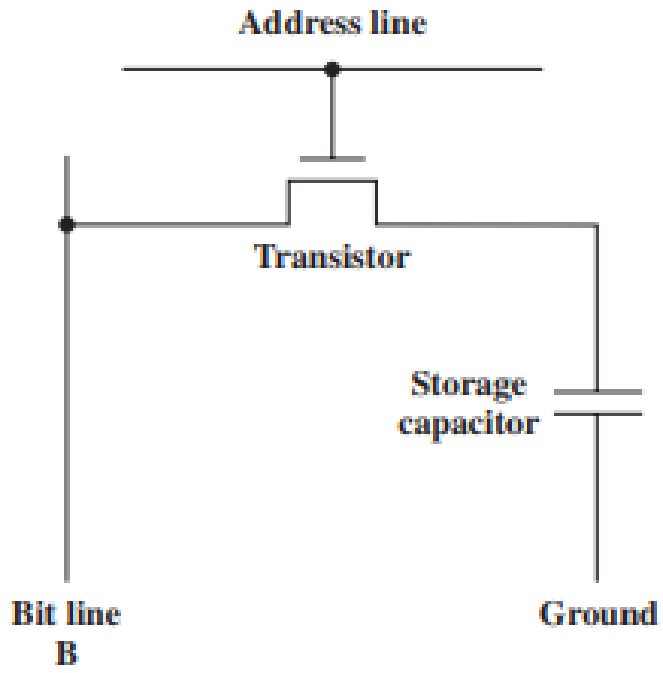


Figure: DRAM cell.

Memory Organization

Semiconductor Memory

- **RAM**
- In a SRAM, binary values are stored using traditional flip-flop logic-gate configurations.
- Unlike the DRAM, no refresh is needed to retain data.
- When a signal is applied to address line, the two transistors are switched on, allowing a read or write operation.
- For write, the desired bit value is applied to line B and its complement to line B bar.
- For read, the bit value is read from line B.
- SRAM is used for cache memory.

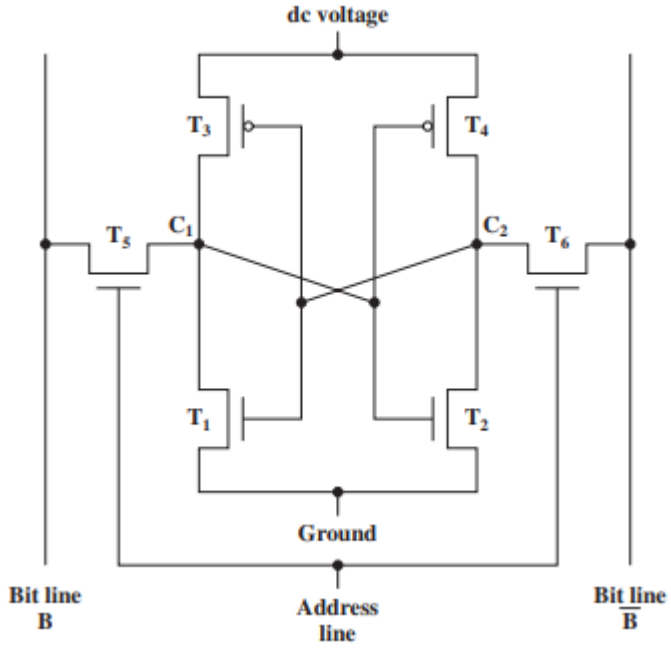


Figure: SRAM cell.

Memory Organization

Semiconductor Memory

ROM

- Data wired into the chip as part of the fabrication process.
- Non-volatile – no power source is required to maintain the bit values in memory.
- It is not possible to write new data into it.

PROM – Programmable ROM

- Non-volatile and may be written into only once.
- Writing process is performed electrically and may be performed at a time later than the original chip fabrication.

Memory Organization

Semiconductor Memory

EPROM – Erasable Programmable ROM

- Read and written electrically.
- Before a write operation, all storage cells must be erased to the same initial state by exposure of the packaged chip to ultraviolet radiation.
- EPROM is more expensive than PROM, but it has the advantage of the multiple update capability.

EEPROM – Electrically Erasable Programmable ROM

- It can be written into at any time without erasing prior contents.
- It combines advantage of nonvolatility with the flexibility of being updatable in place.
- It is more expensive than EPROM and also less dense.

Memory Organization

Semiconductor Memory

Flash Memory

- It is intermediate between EPROM and EEPROM in both cost and functionality.
- It uses an electrical erasing technology.
- It is possible to erase just blocks of memory rather than an entire chip.
- In flash memory, a section of memory cells can be erased in a single action or “flash”.
- It does not provide byte-level erasure.

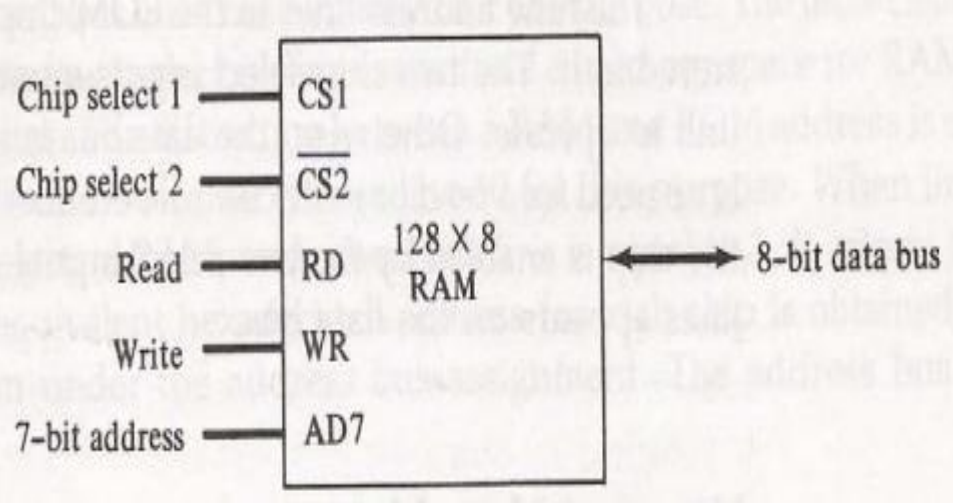
Memory Organization

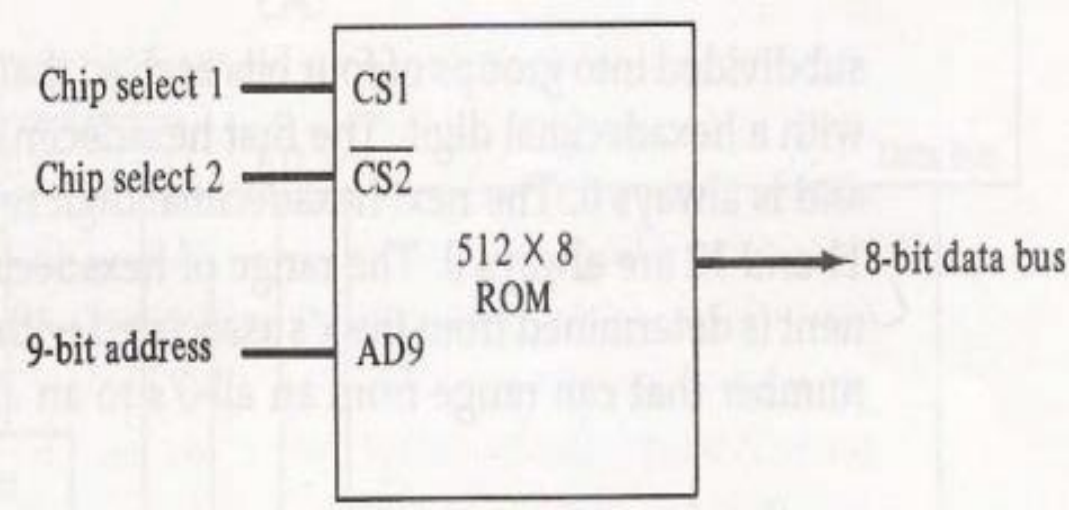
Main Memory

- Most of the main memory is made up of RAM IC chips, but a portion of the memory may be constructed with ROM chips.
- The ROM portion of main memory is needed for storing an initial program called a *bootstrap loader*.
- The bootstrap program loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use.

Memory Organization

Main Memory

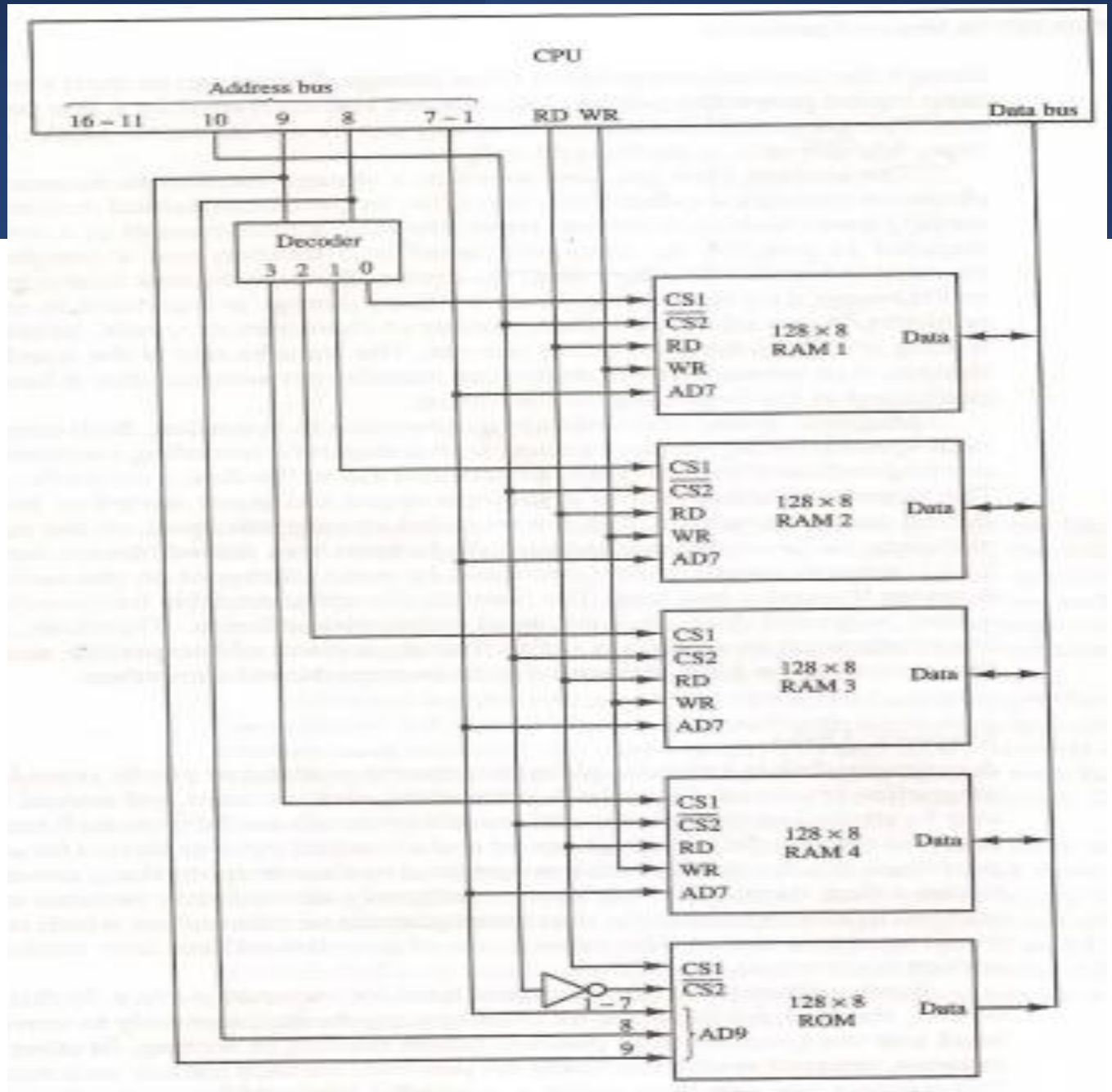




Memory Organization

Main Memory

- Memory connection with CPU



Memory Organization

Auxiliary Memory

- Examples: Magnetic disks, tapes, magnetic drums, and optical disks.
- A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material.
- Both sides of the disk are used and several disks may be stacked on a spindle with read/write heads available on each surface.
- Concentric circles are called **tracks** and tracks are divided into sections called **sectors**.

Memory Organization

Auxiliary Memory

- A disk system is addressed by address bits that specify the disk number, the disk surface, the track number and the sector within the track.
- To make all the records in a sector of equal length, some disks use a variable recording density with higher density on tracks near the center than on tracks near the circumference.

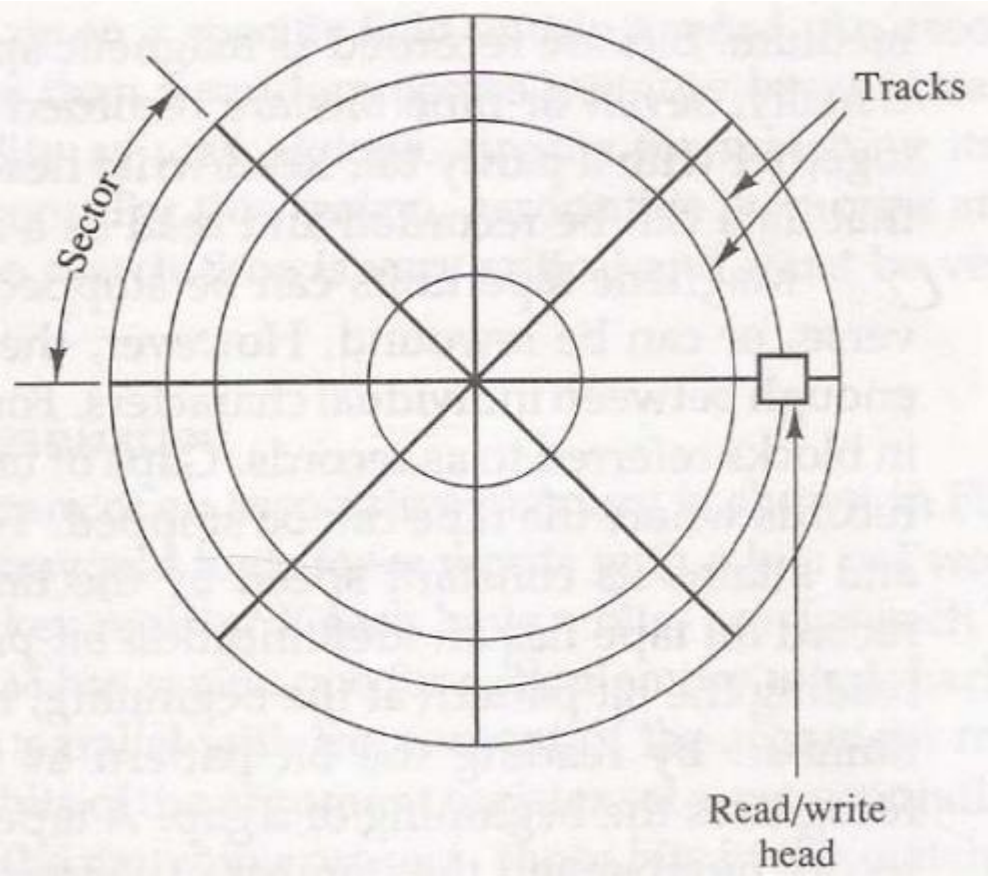


Figure: Magnetic disk.

Memory Organization

Auxiliary Memory

- **Hard disks**: disks that are permanently attached to the unit assembly.
- **Floppy disk**: A disk drive with removable disks.

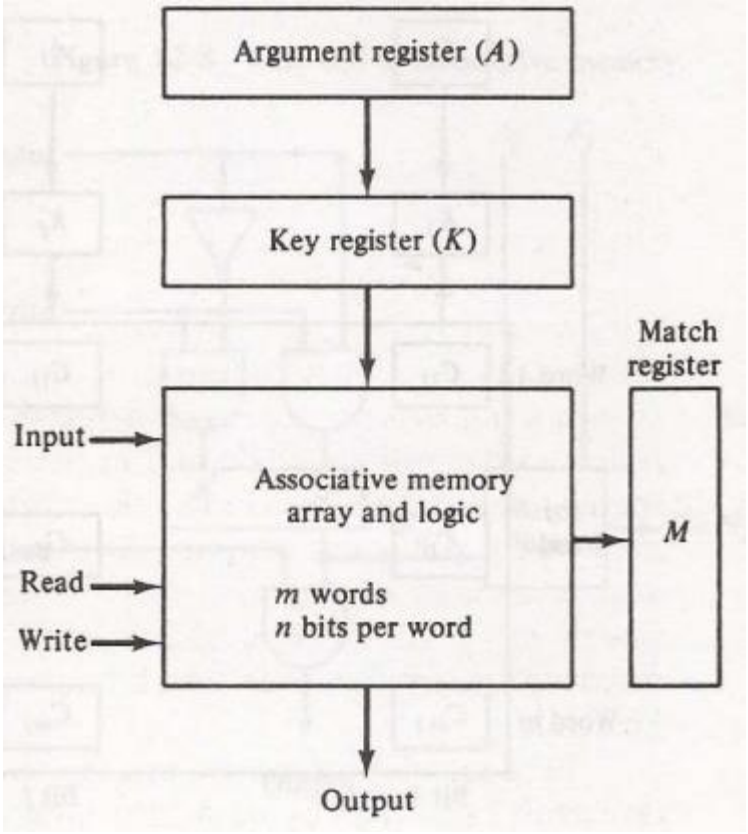
Memory Organization

Associative Memory

- A memory unit accessed by content is called an *associative memory* or *content addressable memory* (CAM).
- When a word is written in an associative memory, no address is given. The memory is capable of finding an empty unused location to store the word.
- When a word is to be read , the content of word, or part od word, is specified. The memory locates all words which match the specified content and marks then for reading.
- Associative memory is more expensive than a random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument.

Memory Organization

Associative Memory

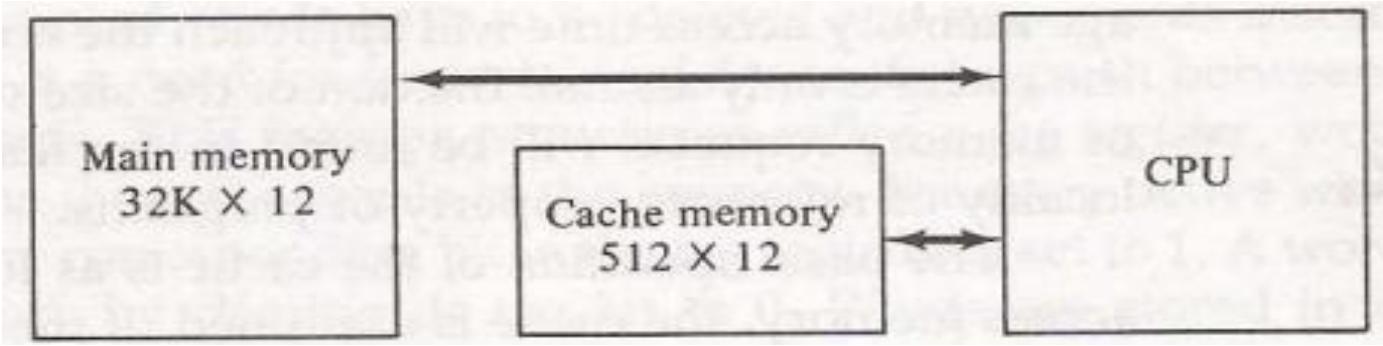


A	101 111100	
K	111 000000	
Word 1	100 111100	no match
Word 2	101 000001	match

Memory Organization

Cache Memory

- Fast, small memory.
- The active (most frequently accessed) portions of the program and data are placed in cache memory to reduce the average memory access time and total execution time of the program.
- It is placed between the CPU and main memory.



Memory Organization

Cache Memory

- The cache memory access time is less than the access time of main memory by a factor of 5 to 10.
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.
- A block of words is then transferred from main memory to cache memory. The block size may vary from one word to 16 words adjacent to the one just accessed.
- The performance of cache memory is frequently measured in terms of ***hit ratio***.
- The transformation of data from main memory to cache memory is referred to as a **mapping process**.

Memory Organization

Cache Memory

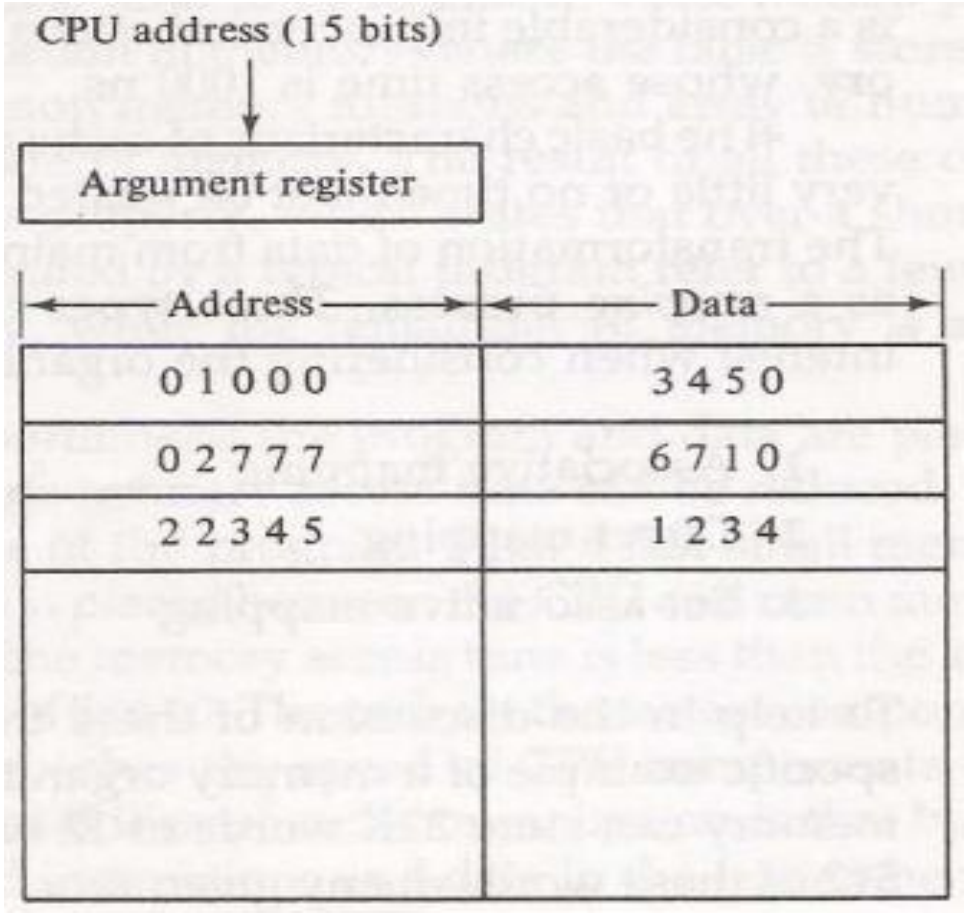
- Mapping techniques:
  1. Associative mapping
  2. Direct mapping
  3. Set-associative mapping

Memory Organization

Cache Memory

1. **Associative mapping cache**

- Any location in cache can store any word from main memory.
- A CPU address is placed in argument register and the associative memory is searched for a matching address.



Memory Organization

Cache Memory

1. **Associative mapping cache**

- If address is found, the corresponding 12-bit data is read and sent to CPU. Otherwise, main memory is accessed for the word and address-data pair is then transferred to the cache.
- If the cache is full, an address-data pair is displaced using replacement algorithm to make room for a pair that is needed and not presently in the cache.

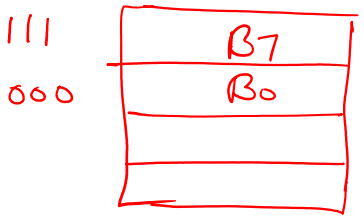
Memory Organization

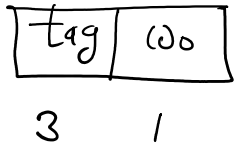
Associative Mapping:

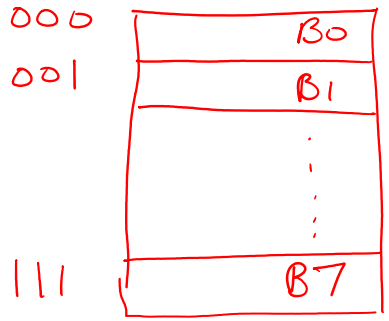
Example :

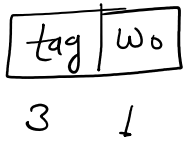
MM: Main Memory

MM Block	Associative Mapping	CM Line
B7 [111]	No mapping function →	Any line
B0 [000]	No mapping function →	Any line









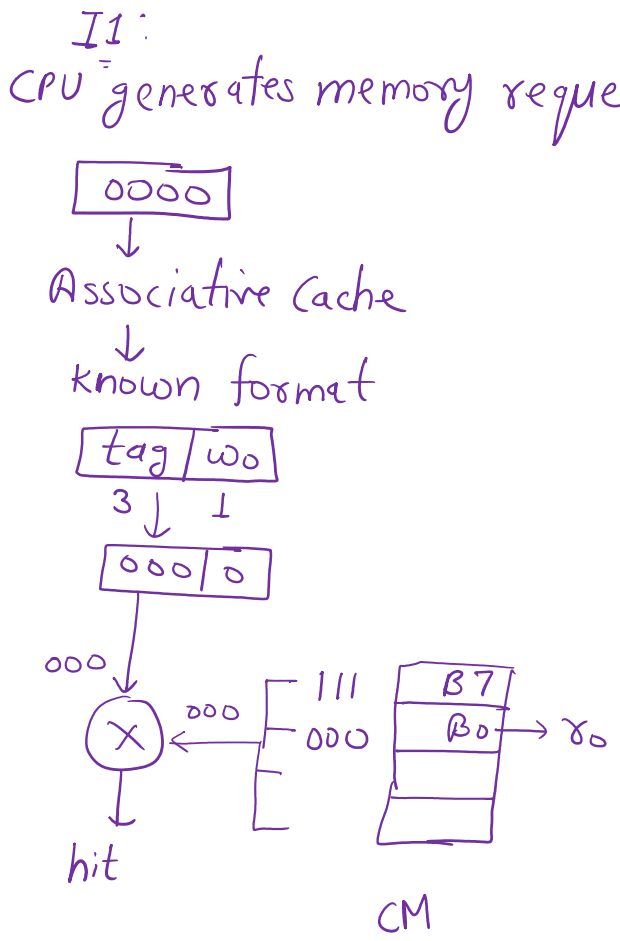
Memory Organization

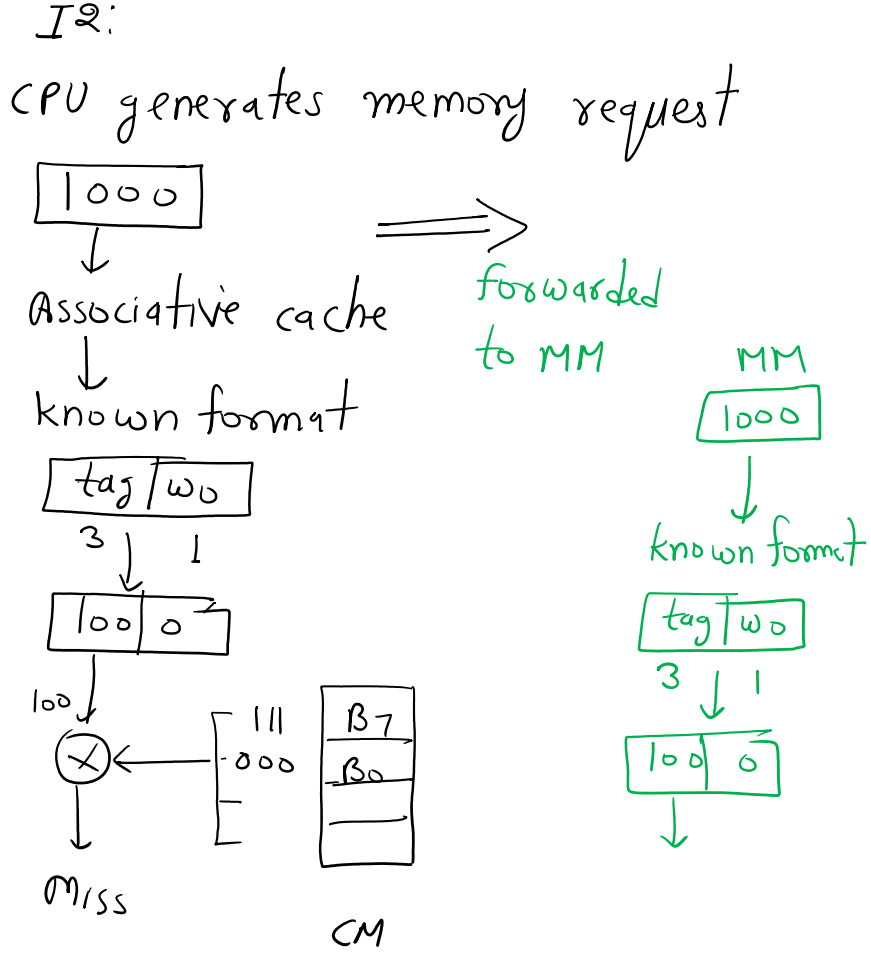
Associative Mapping:

I1 : MOV r_0, [0000]

I2: MOV r_1, [1000]

I3 : ADD r_0, r_1





Memory Organization

Associative Mapping:

- The CPU generated memory request is initially referred to the associative cache.
- The associative cache controller interprets the request into its known format i.e. tag | w_o
- The CPU generated tag is compared with the existing tags in the cache controller.
- If any matched tag is there then operation is *hit;* and based on the word offset data is transferred to the CPU.
- If no matching tag is there then operation is *miss*; and reference is forwarded to main memory.
- The MM controller interprets the requests into its known format tag | w_o and corresponding MM block is transferred to the CM by using the associative mapping technique. Later data is transferred to CPU.

* Tag memory size = number of cache lines * number of tag bits in each line
= 4 * 3 = 12 bits

Memory Organization

Direct Mapping:

- In this technique, mapping function is used to transfer the data from MM to CM.
- The mapping function is: $i = k \bmod n$

i: line number in CM

k: MM block number

n: number of lines in CM

$0 \leftarrow 0 \bmod 4$

$1 \leftarrow 1 \bmod 4$

$2 \leftarrow 2 \bmod 4$

$3 \leftarrow 3 \bmod 4$

$0 \leftarrow 4 \bmod 4$

$1 \leftarrow 5 \bmod 4$

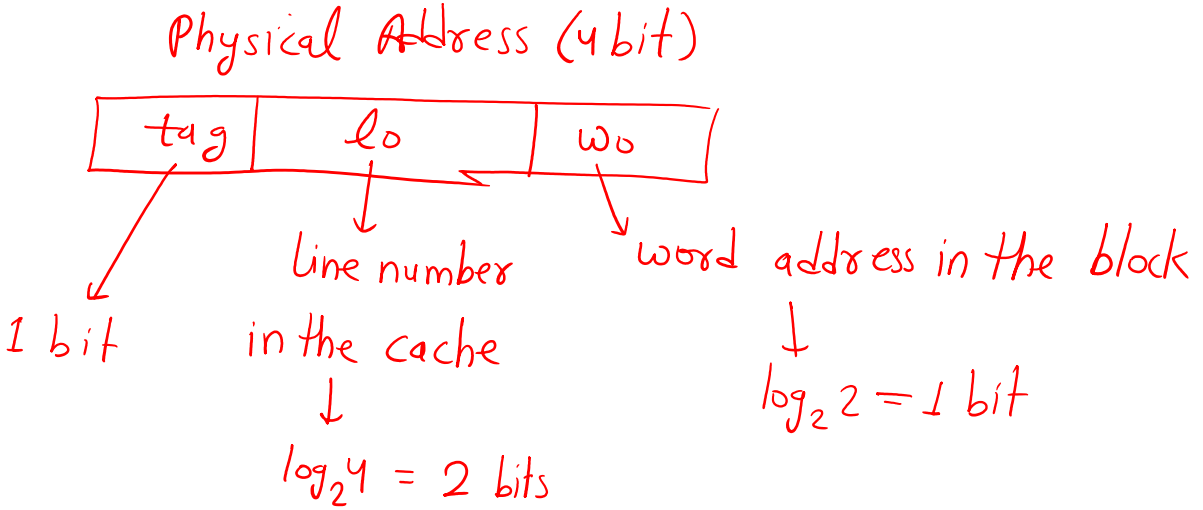
$2 \leftarrow 6 \bmod 4$

$3 \leftarrow 7 \bmod 4$

Memory Organization

Direct Mapping:

- The Direct Cache Controller interprets the CPU generated request as:



Memory Organization

Direct Mapping:

- Example:

MM block → Direct Mapping → CM Line

B0 → k mod n → 0 mod 4 → Line 0

B7 → " → 7 mod 4 → Line 3

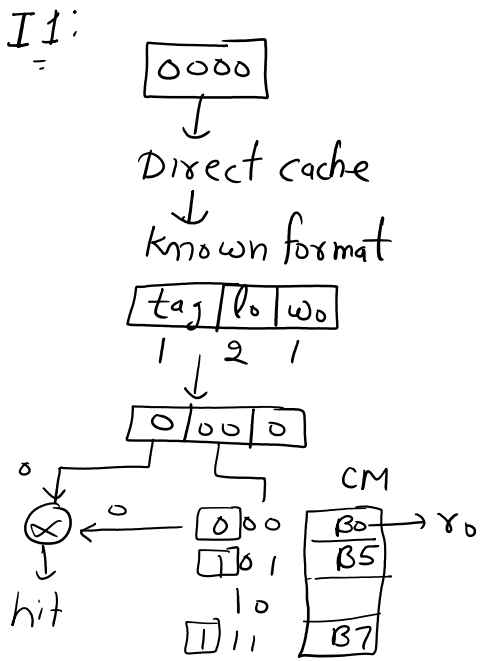
B5 → " → 5 mod 4 → Line 1

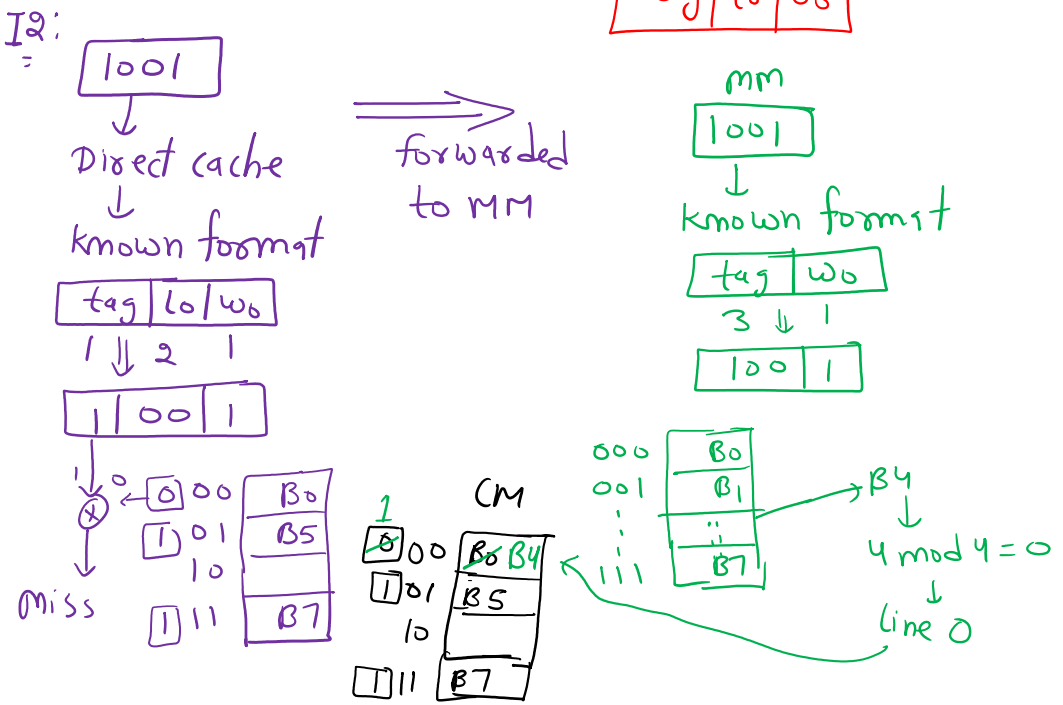
Consider:

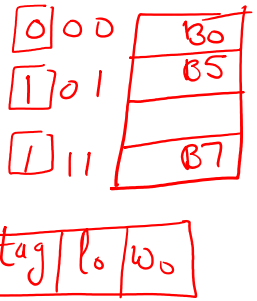
I1: Mov r_0, [0000]

I2: Mov r_1, [1001]

I3: Add r_0, r_1







Memory Organization

Direct Mapping:

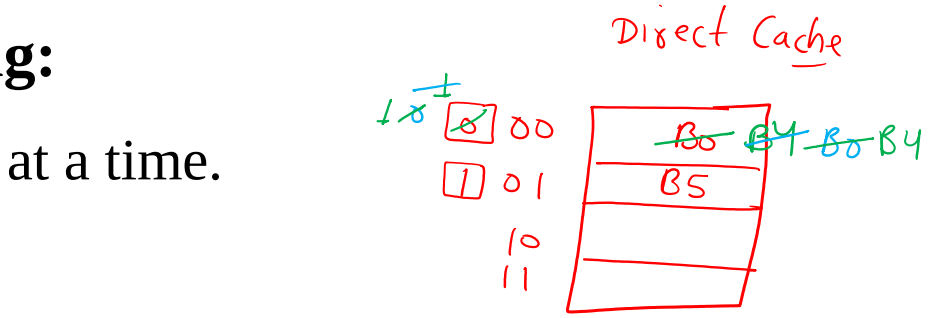
- Direct Cache Controller interprets the request into its known format i.e. tag | lo | wo
- The line offset is mapped with the address logic of the cache memory to enable the corresponding line.
- The existing tag in the enabled line is compared with the CPU generated tag.
- When both tags are matched, operation is called *hit* and data get transferred to CPU based on word offset. Otherwise, operation is *miss* and reference is forwarded to main memory.
- The required block is transferred from MM to CM based on Direct Mapping function.
- Tag memory size = number of cache line * number of tag bits in each line.

= 4 * 1 bits

Memory Organization

Disadvantage of Direct Mapping:

- Each line can hold only one block at a time.
- Conflict misses are high.



To minimize conflict misses → solution → set-associative cache

Memory Organization

Set-Associative Mapping:

- In this technique, mapping function is used to transfer the data from MM to CM.
- The mapping function is: $i = k \bmod s$

i : set number in CM

k : MM block number

s : number of sets in CM

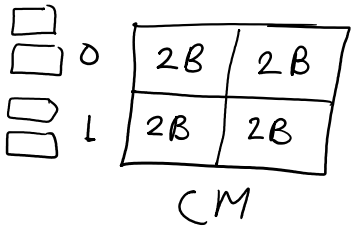
$s = \frac{N}{p\text{-way}}$ → N → number of cache lines

p-way → number of blocks placed at a time in each cache line.

* Considering:
[Block size = 2 bytes]
CM = 8B, MM = 16B

$0 \leftarrow 0 \bmod 2$
$1 \leftarrow 1 \bmod 2$
$0 \leftarrow 2 \bmod 2$
$1 \leftarrow 3 \bmod 2$
$0 \leftarrow 4 \bmod 2$
$1 \leftarrow 5 \bmod 2$
$0 \leftarrow 6 \bmod 2$
$1 \leftarrow 7 \bmod 2$

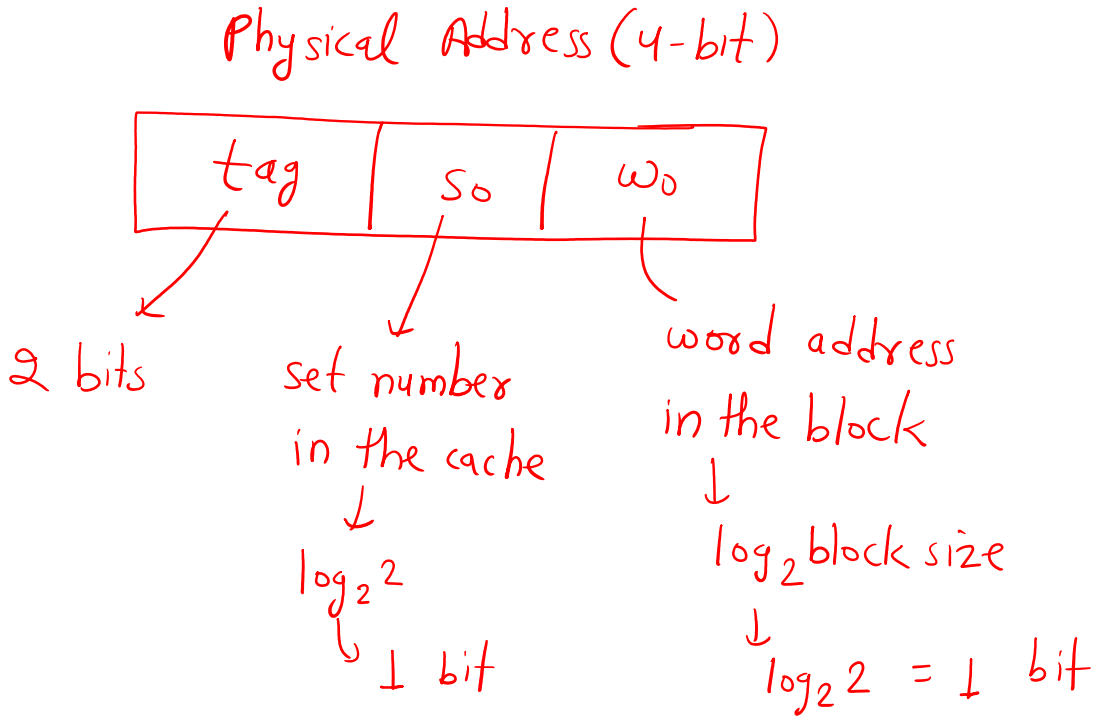
2-way set associative.



Memory Organization

Set-Associative Mapping:

- The Set-Associative Cache Controller interprets the CPU generated request as:



Memory Organization

Set-Associative Mapping:

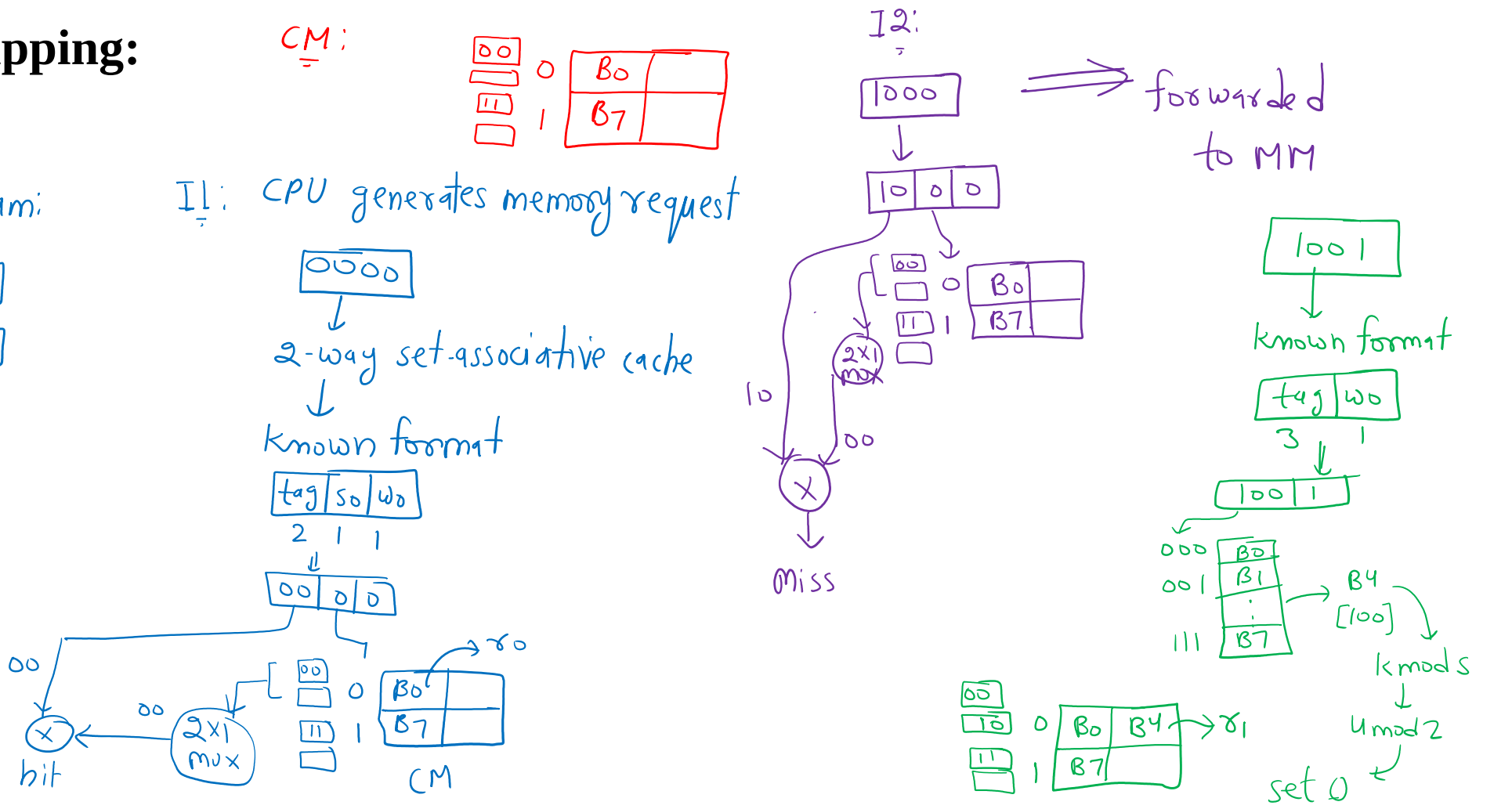
- Example:

Consider this program:

I1: mov r0, [0000]

I2: mov r1, [1001]

I3: Add r0, r1



Memory Organization

Set-Associative Mapping:

- Set-Associative Cache Controller interprets the CPU generated request as:
- The set offset field is mapped with the address logic of the cache memory to enable the corresponding cache set.
- Each set contains multiple tags. To compare the existing tags of the enabled set with the CPU generated tag, a multiplexer is used.
- If any tag is matched, operation is called *hit* and data get transferred to CPU based on word offset. Otherwise, operation is *miss* and reference is forwarded to main memory.
- The required block is transferred from MM to CM based on Set-Associative Mapping function.
- Tag memory size = number of sets * number of blocks in each set * number of tag bits.

= 2 * 2 * 2 ⇒ 8 bits

Memory Organization

Replacement Algorithms: FIFO, LRU (Least Recently Used)

- Example: Consider 4 block CM (initially empty) with the following main memory block references: 4, 5, 7, 12, 4, 5, 13, 4, 5, 7. Identify the hit ratio by using: FIFO, LRU, Direct mapped cache, 2-way set associative with LRU.