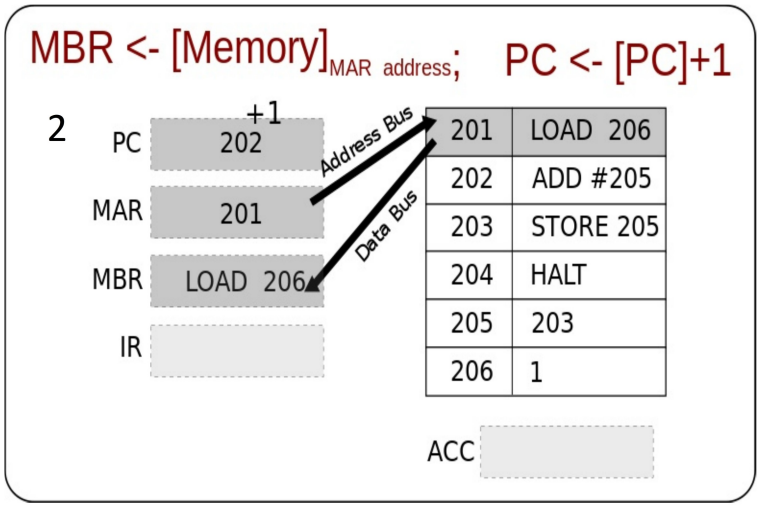
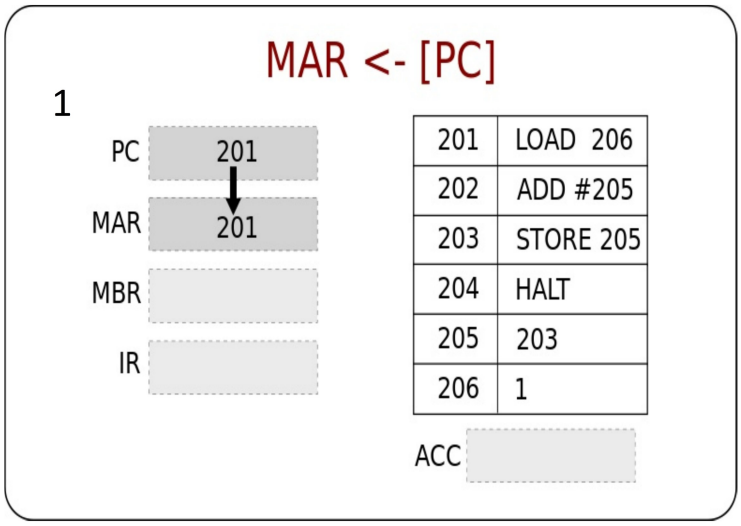
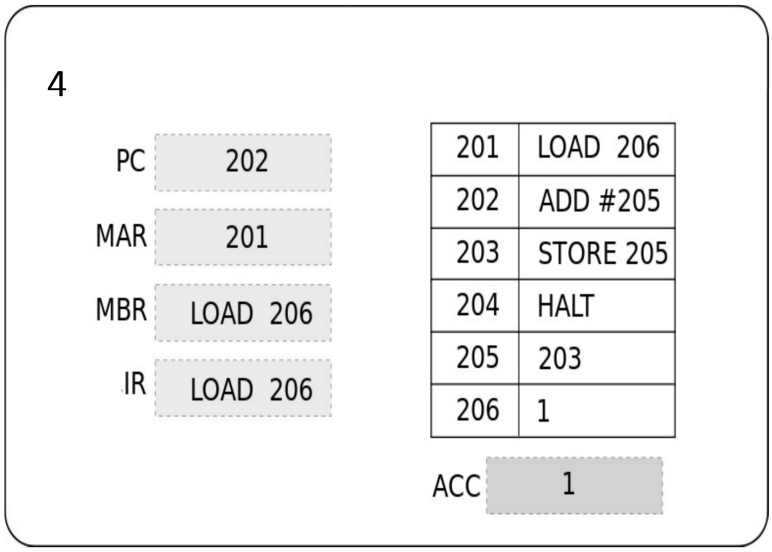
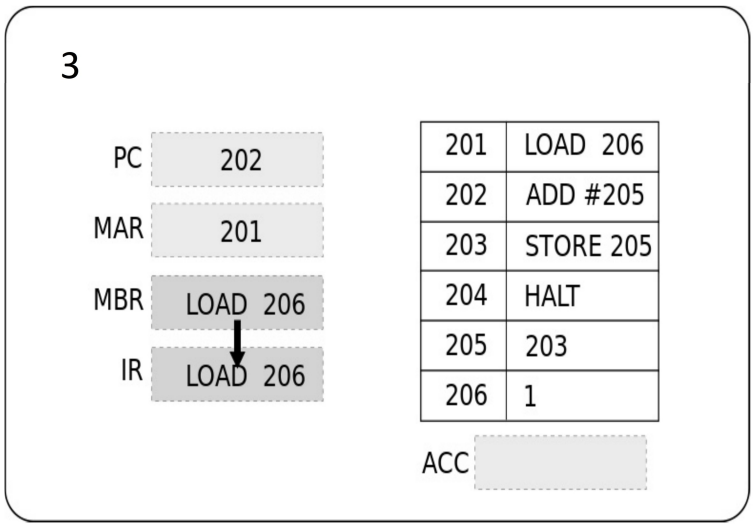

MBR <- [Memory]MAR address; PC <- [PC]+1
2
+1
PC
202
MAR
201
MBR
LOAD 206
IR
Address Bus
Data Bus
201 LOAD 206
202 ADD #205
203 STORE 205
204 HALT
205 203
206 1
ACC

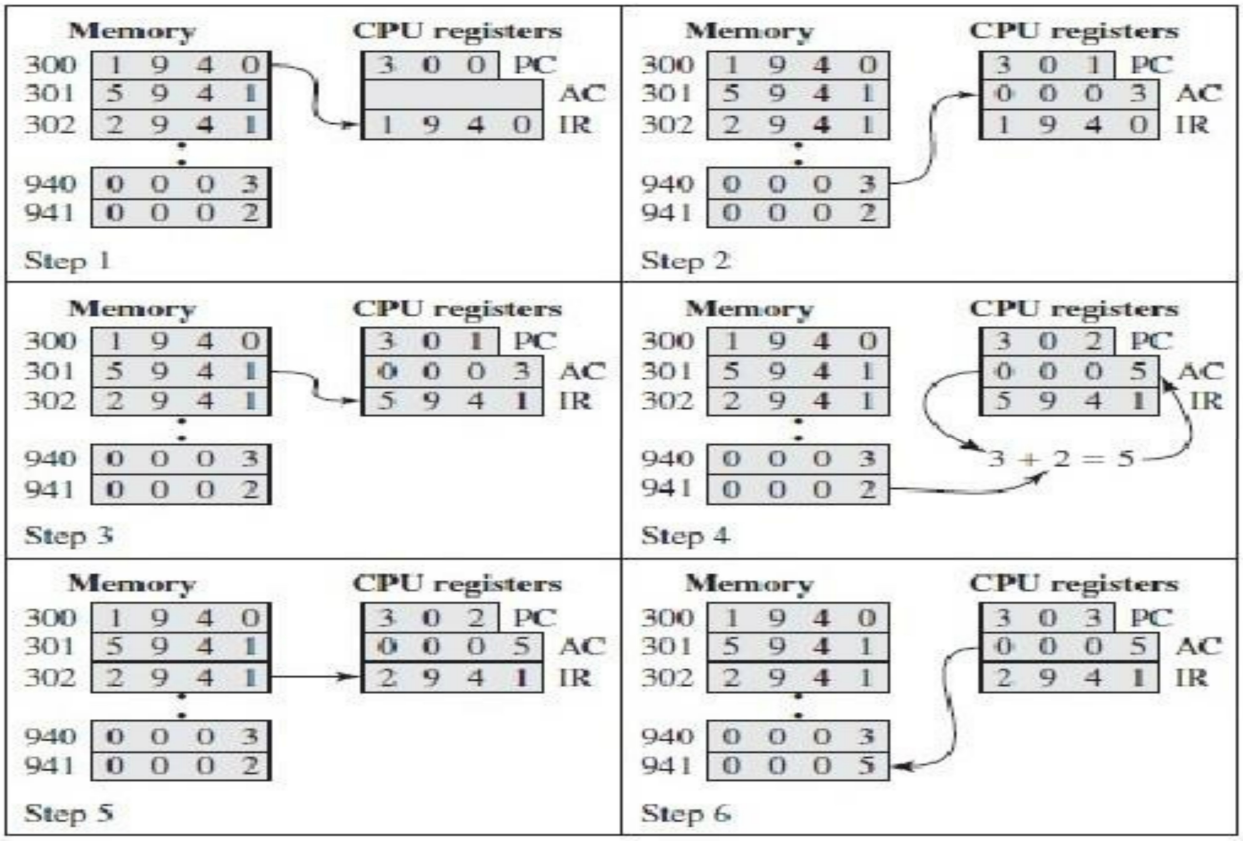

MAR <- [PC]
1
PC
201
MAR
201
MBR
IR
201 LOAD 206
202 ADD #205
203 STORE 205
204 HALT
205 203
206 1
ACC


4
PC
202
MAR
201
MBR
LOAD 206
IR
LOAD 206
201 LOAD 206
202 ADD #205
203 STORE 205
204 HALT
205 203
206 1
ACC
1


3
PC
202
MAR
201
MBR
LOAD 206
IR
LOAD 206
201 LOAD 206
202 ADD #205
203 STORE 205
204 HALT
205 203
206 1
ACC

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory



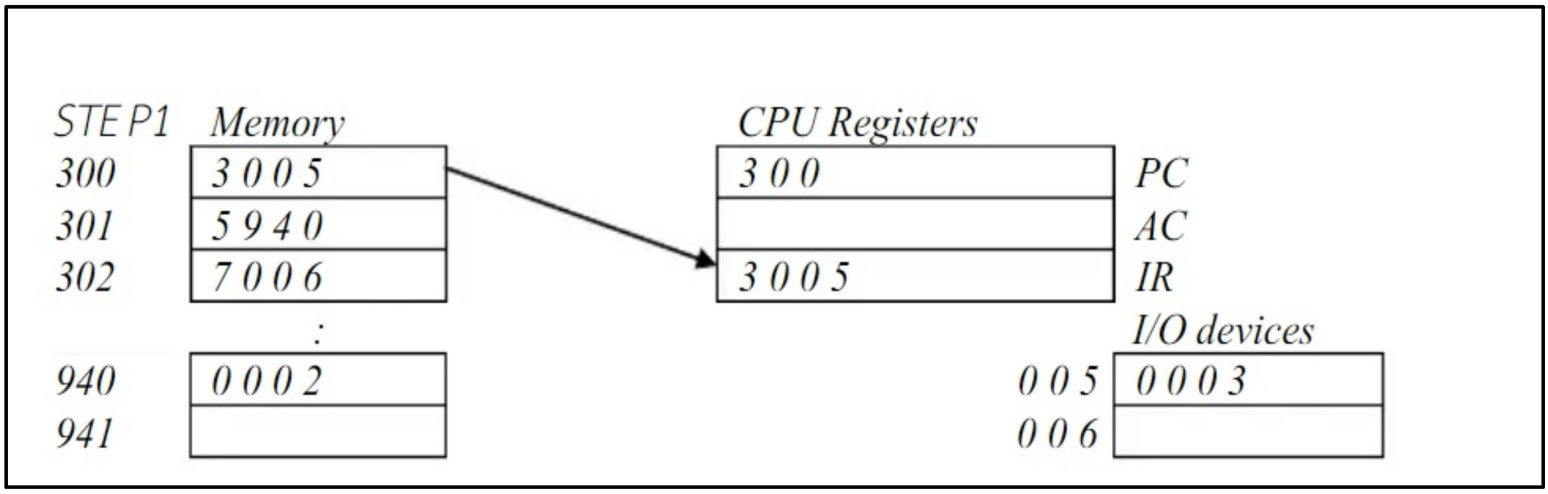
Example

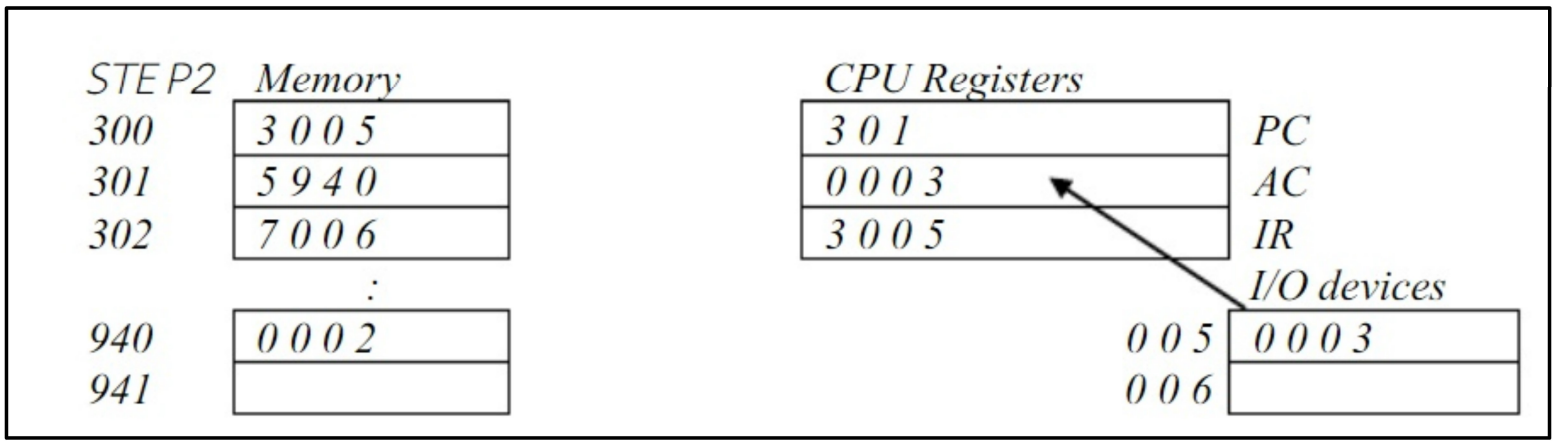
Example

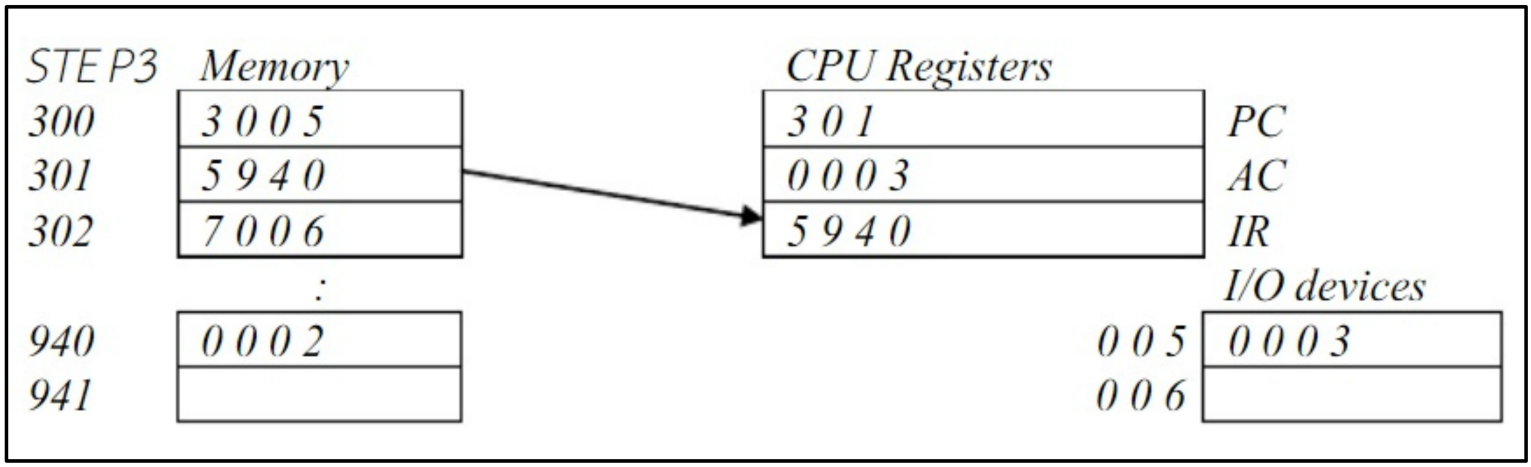
- The PC contains the address of the first instruction (300) which value in hexadecimal is (1940) that is loaded into IR and the PC is incremented.
- The first hexadecimal digit in the IR indicate that the AC is to be loaded. The remaining 3 hexadecimal digits represent the address (940) from which data are to be loaded.
- The next instruction (5941) is fetched from location 301 and the PC is incremented.
- The old contents of the AC and the contents of 941 are added and the result is stored in the AC.
- The next instruction (2941) is fetched from 302 and the PC is incremented.
- The contents of the AC are stored in 941.

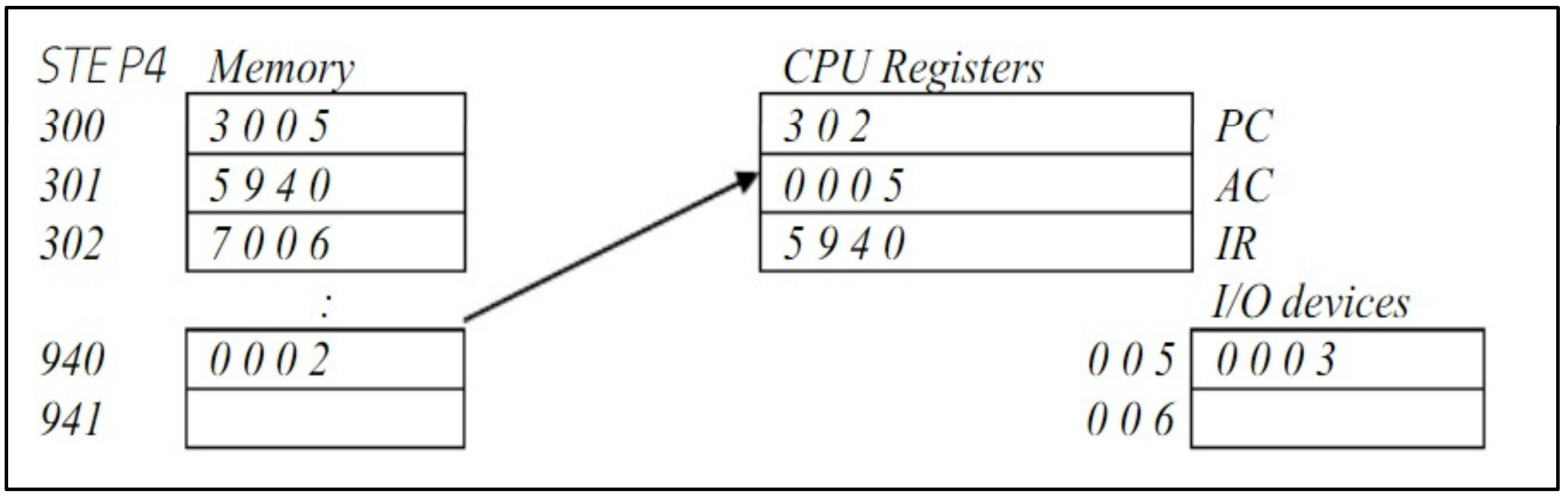
Assignment

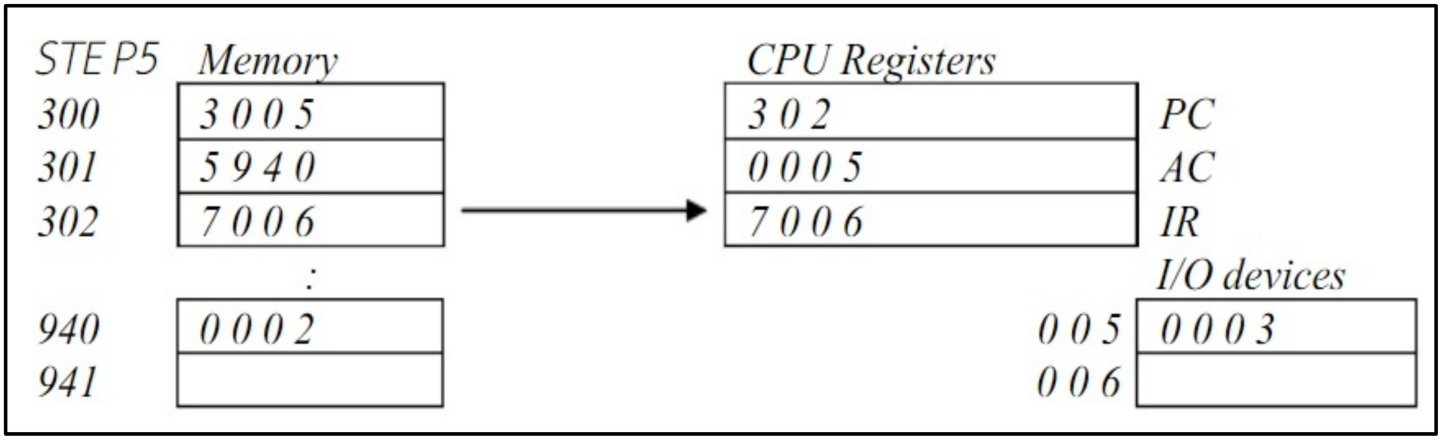
- Assume that the hypothetical machine in the program execution figure also has two I/O instructions:
- 0011 Load AC from I/O
- 0111 Store AC to I/O
- Where, in these cases, the 12-bit address identifies a particular I/O device. Show the program execution for the following program:
    - Load AC from device 5.
    - Add contents of memory location 940.
    - Store AC to device 6.
- Assume that the next value retrieved from device 5 is 3 and that location 940 contains a value of 2.

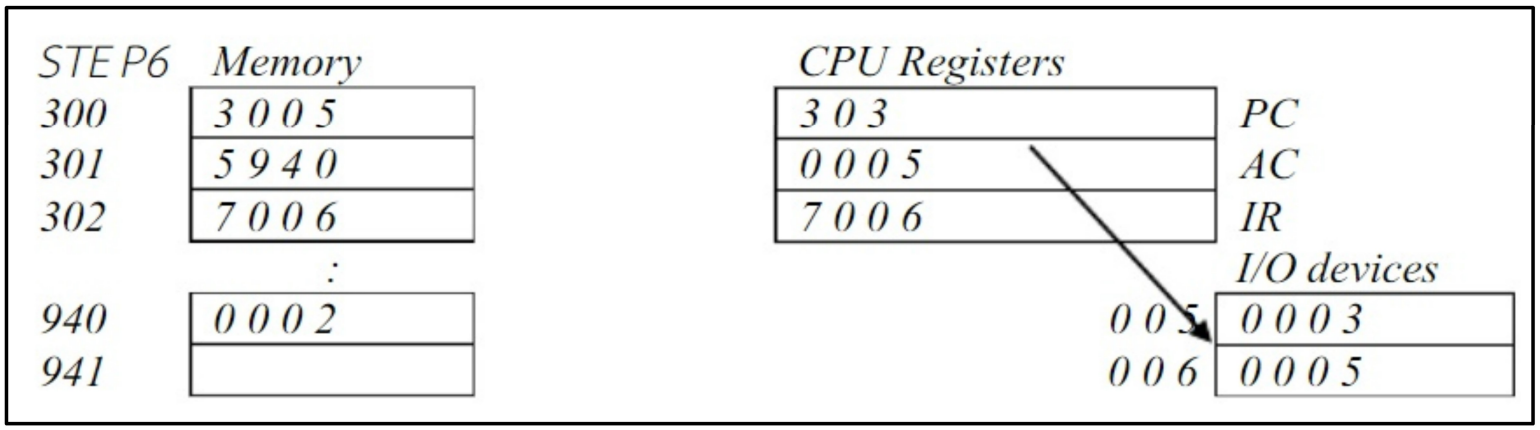

STEP1
Memory
CPU Registers
300
3 0 0 5
301
5 9 4 0
302
7 0 0 6
:
940
0 0 0 2
941
3 0 0
PC
AC
3 0 0 5
IR
I/O devices
0 0 5
0 0 0 3
0 0 6


STEP2
Memory
CPU Registers
300
3 0 0 5
301
5 9 4 0
302
7 0 0 6
:
940
0 0 0 2
941
3 0 1
PC
0 0 0 3
AC
3 0 0 5
IR
I/O devices
0 0 5
0 0 0 3
0 0 6


STEP3
Memory
CPU Registers
300
3 0 0 5
301
5 9 4 0
302
7 0 0 6
:
940
0 0 0 2
941
3 0 1
PC
0 0 0 3
AC
5 9 4 0
IR
I/O devices
0 0 5
0 0 0 3
0 0 6

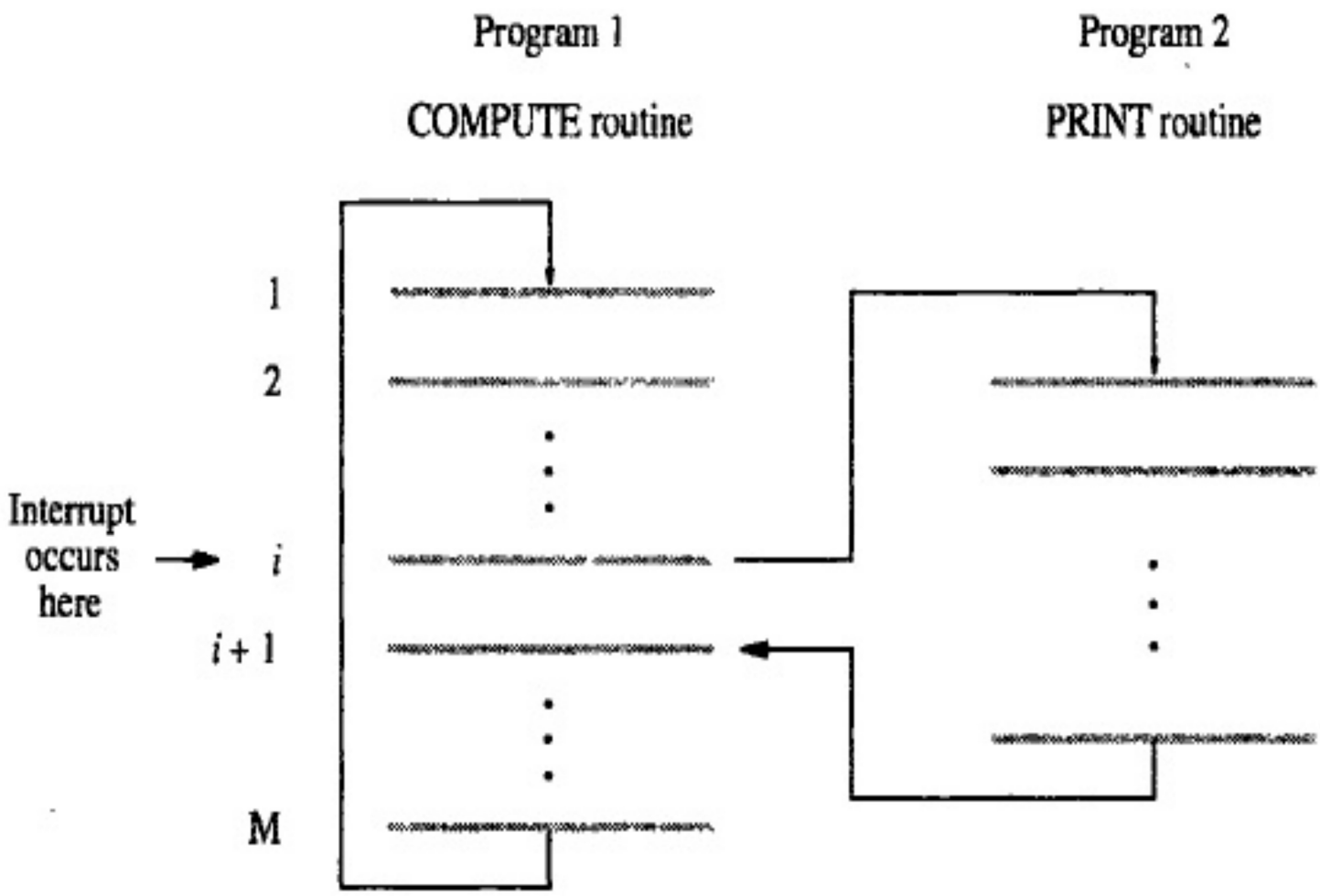

STEP4
Memory
CPU Registers
300
3 0 0 5
301
5 9 4 0
302
7 0 0 6
:
940
0 0 0 2
941
3 0 2
PC
0 0 0 5
AC
5 9 4 0
IR
I/O devices
0 0 5
0 0 0 3
0 0 6


STE P5
Memory
300
3 0 0 5
301
5 9 4 0
302
7 0 0 6
:
940
0 0 0 2
941
CPU Registers
3 0 2
PC
0 0 0 5
AC
7 0 0 6
IR
I/O devices
0 0 5
0 0 0 3
0 0 6


STE P6
Memory
300
3 0 0 5
301
5 9 4 0
302
7 0 0 6
:
940
0 0 0 2
941
CPU Registers
3 0 3
PC
0 0 0 5
AC
7 0 0 6
IR
I/O devices
0 0 5
0 0 0 3
0 0 6
0 0 0 5

Interrupting the instruction execution cycle

- Interrupt : is an external event to the currently executing process that causes a change in the normal flow of instruction execution.
- An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.
- The processor suspend its current activities, saving its state, and executing a function called interrupts handler or interrupt service routine,
- After the interrupt handler finishes, the processor resumes execution of the previous thread.


Program 1
COMPUTE routine
Program 2
PRINT routine
1
2
Interrupt
occurs
here
i
i + 1
M

Interrupt Types

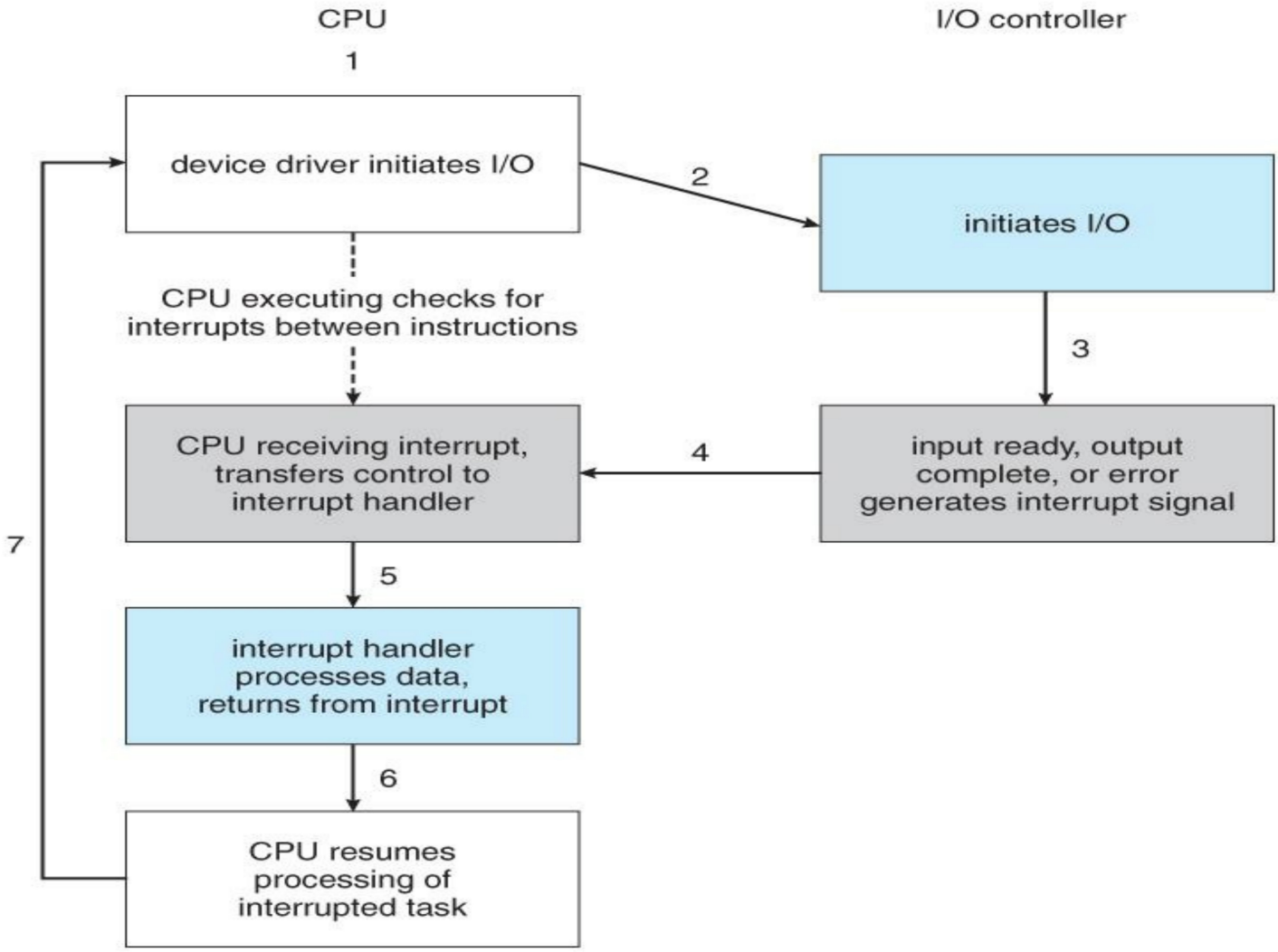
- **External Interrupts**
  - come from external input / output devices which are connected externally to the processor.
- **Internal Interrupts**
  - known as traps
  - Caused by
    - some illegal operation
    - erroneous use of data.
    - exception that has been caused by the program itself.
    - E.G. : division by zero, processor timer or an invalid op-code ……
- **Software interrupts**
  - occur only during the execution of an instruction.
    - E.G : switch from user mode to supervisor mode.

Interrupt Execution Cycle

1. I/O device issues the interrupt.
2. The processer finishes the execution of an instruction.
3. The processor checks for an interrupt. If there is one, it then sends an acknowledgement signal to the I/O device that issued the interrupt. This signal allows the device to remove its interrupt signal.
4. To switch to run interrupt handler, information about the current program is stored, so that its execution may be resumed later, including PC.
5. The processor loads the PC with the entry location of the interrupt handler.

Interrupt Execution Cycle

6. A typical case is there are a set of routines, each for one type of interrupt, or each for one device, through the interrupt vector, which contains the addresses of all the service routines.
7. The interrupt handler may continue to save other information that is considered as part of process state.
8. The handler performs the interrupt processing.
9. When the handler finishes, the saved register values are restored into the registers that originally hold them when the interrupt handler returns.
10. Finally PC values of the interrupted program are restored, thus the program may continue to execute.


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

Multiple Interrupts

- If multiple interrupts have occurred how the processor does decide which one to process?.
- **Multiple Interrupt Lines**
  - Processor picks the interrupt line with highest priority.
  - Priorities are assigned to interrupt lines during the processor design phase
- **Software Poll**
  - processor detects an interrupt
  - branches to ISR to poll each I/O module to determine which module caused the interrupt.
  - processor raises TESTI/O and place the address of a particular I/O module on the address lines.
  - I/O module responds positively if it set the interrupt.
  - With software polling, the order in which modules are polled determines their priority.

Multiple Interrupts

- **Daisy Chain**
  - I/O modules share a common interrupt request lines.
  - Interrupt acknowledge line is connected in daisy sends out an interrupt acknowledge.
  - The interrupt acknowledge signal propagates through a series of I/O module until it gets to a requesting module.
  - The requesting module typically responds by placing a word (vector) on the data lines.
  - The vector is either the address of the I/O module or some other unique identification.
  - priority of a module is determined by the position of the module in the daisy chain
  - the processor uses the vector as a pointer to the appropriate device service routine

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Multiple Interrupts

- **Bus Arbitration :** In case of bus arbitration method, more than one module may need control of the bus, so some method of arbitration is needed.
- The methods of arbitration can be classified into two groups:
  - **Centralized Scheme**
    - a single hardware device, referred to as a bus controller or arbiter is responsible for allocating time on the bus.
    - The device may be a separate module or part of the processor.
  - **Distributed Scheme**
    - there is no control controller.
    - each module contains access control logic and the modules act together share bus.
    - It is also possible to combine different device identification techniques to identify the devices and to set the priorities of the devices.