William Stallings Computer Organization and Architecture

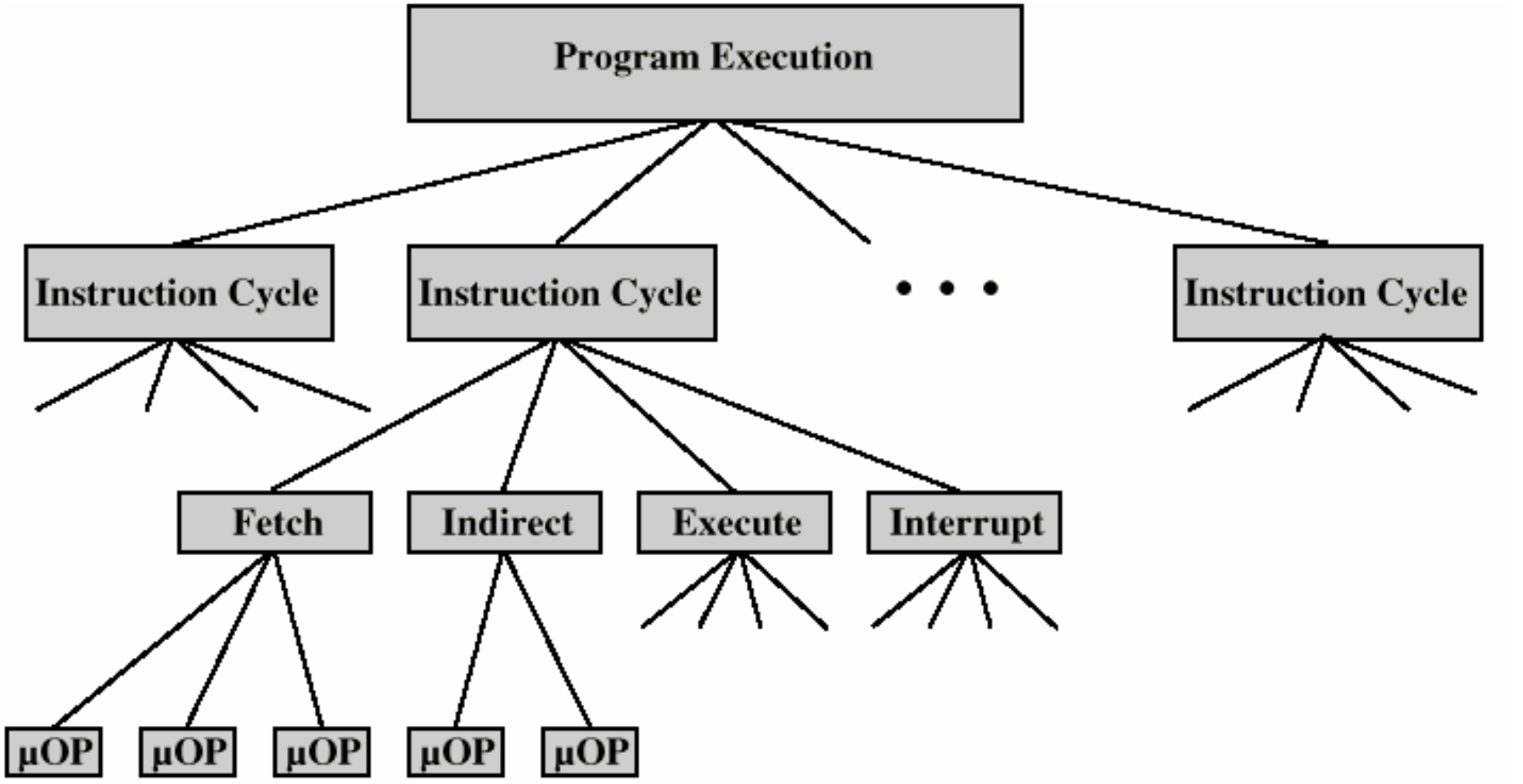
Chapter 14

Control Unit Operations

Execution of the Instruction Cycle

- It has many elementary phases, each executed in a single clock cycle (remember pipelining)
- In each phase only very simple operations (called **micro-operations**) are executed:
  - Move contents between registers (internals, interface with ALU, interface with memory)
  - Activate devices (ALU, memory)
- Micro-operations are the CPU atomic operations, hence define its low-level behaviour
- A micro-operation is the set of actions (data flows and controls) that can be completed in a single clock cycle

Constituent Elements of Program Execution



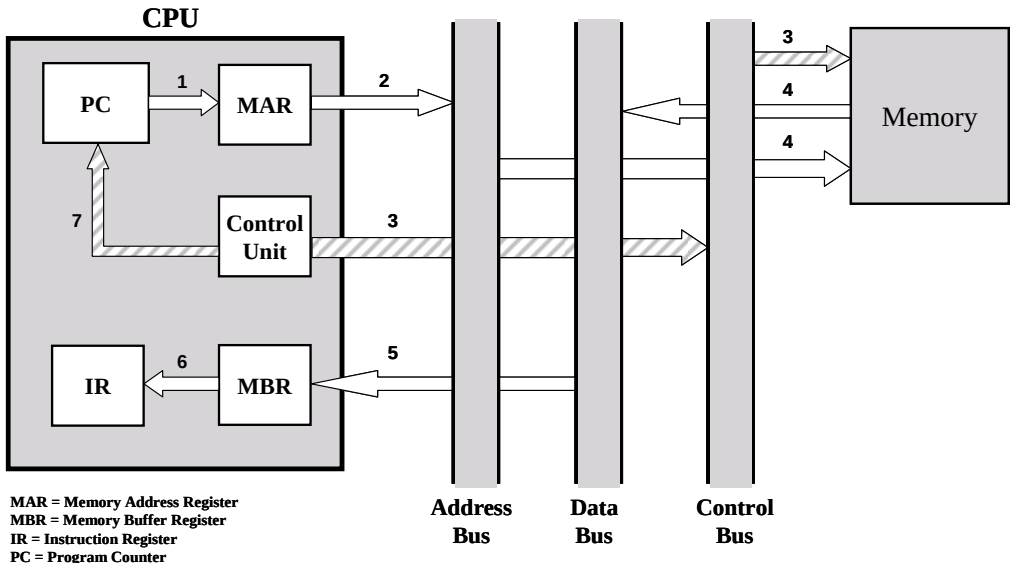
Sequence of micro-operations for instruction fetch

- t_1: MAR <- (PC) <DF_1 >
- t_2: MBR <- (memory) <DF_2 DF_3 DF_4 DF_5 >
  PC <- (PC) +1 <DF_7 >
- t_3: IR <- (MBR) <DF_6 >

(each t_i is a clock cycle, i.e. an atomic time unit)

An alternative organization

- t_1: MAR <- (PC)
- t_2: MBR <- (memory)
- t_3: PC <- (PC) +1
  IR <- (MBR)

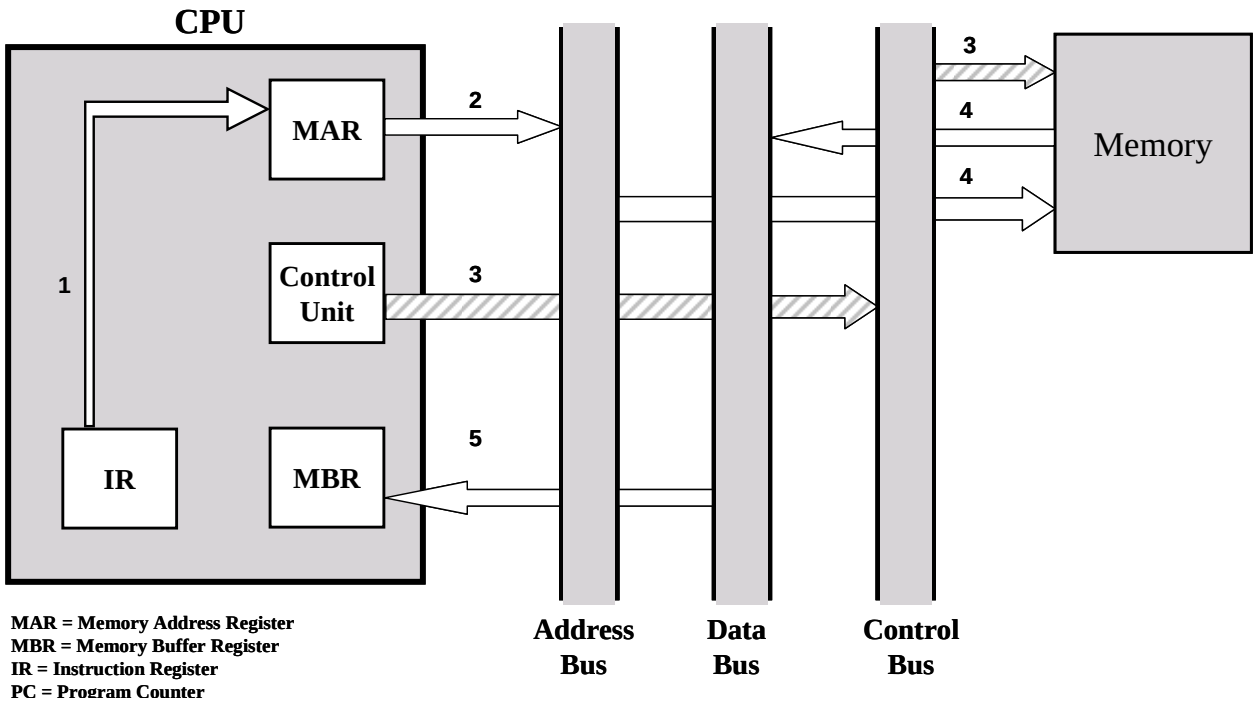


Rules for micro-operation sequencing

- Proper precedence must be observed
  - MAR <- (PC) must precede MBR <- (memory)
- Conflicts must be avoided
  - Must not read & write same register at same time
  - MBR <- (memory) & IR <- (MBR) must not be in same cycle
- Also: PC <- (PC) +1 involves addition
  - Depending on the kind of ALU may need additional micro-operations, hence it may be better to have it in t_2
- Minimization of the number of micro-operations is an algorithmic problem on graphs

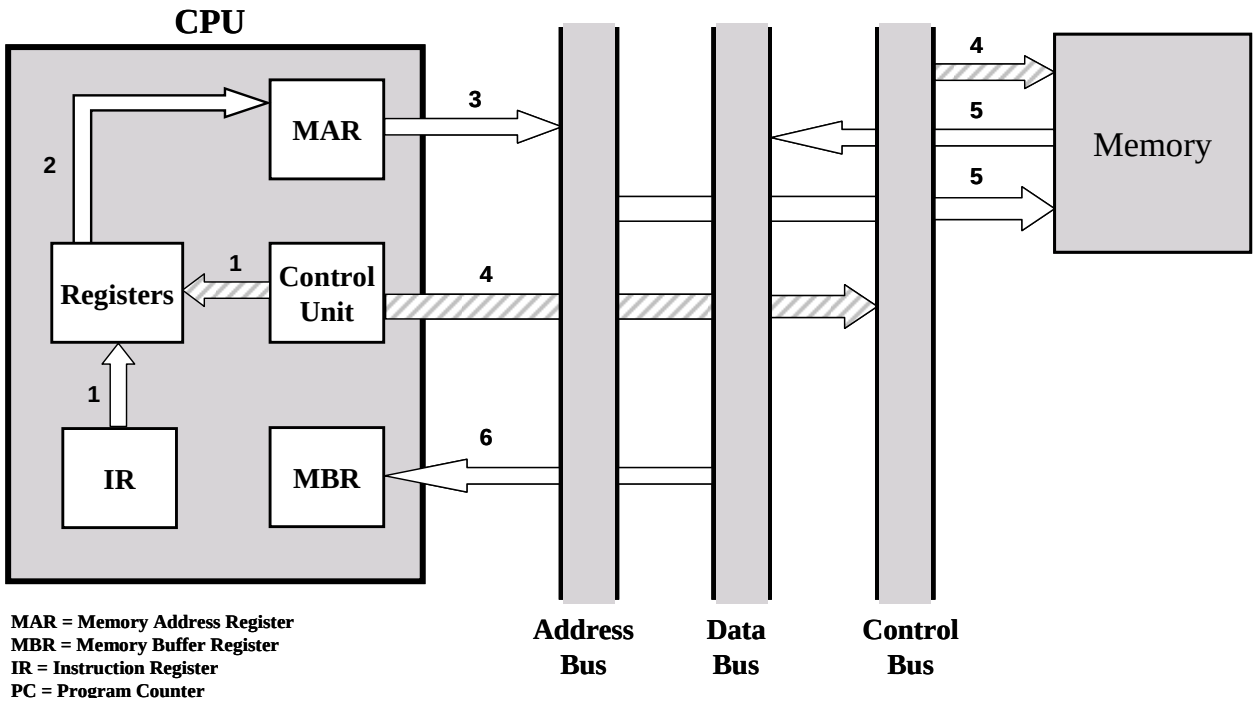
Sequence of micro-operations for direct addressing

- t_1: MAR <- (IR) $<DF_1>$
- t_2: MBR <- (memory) $<DF_2\ DF_3\ DF_4\ DF_5>$



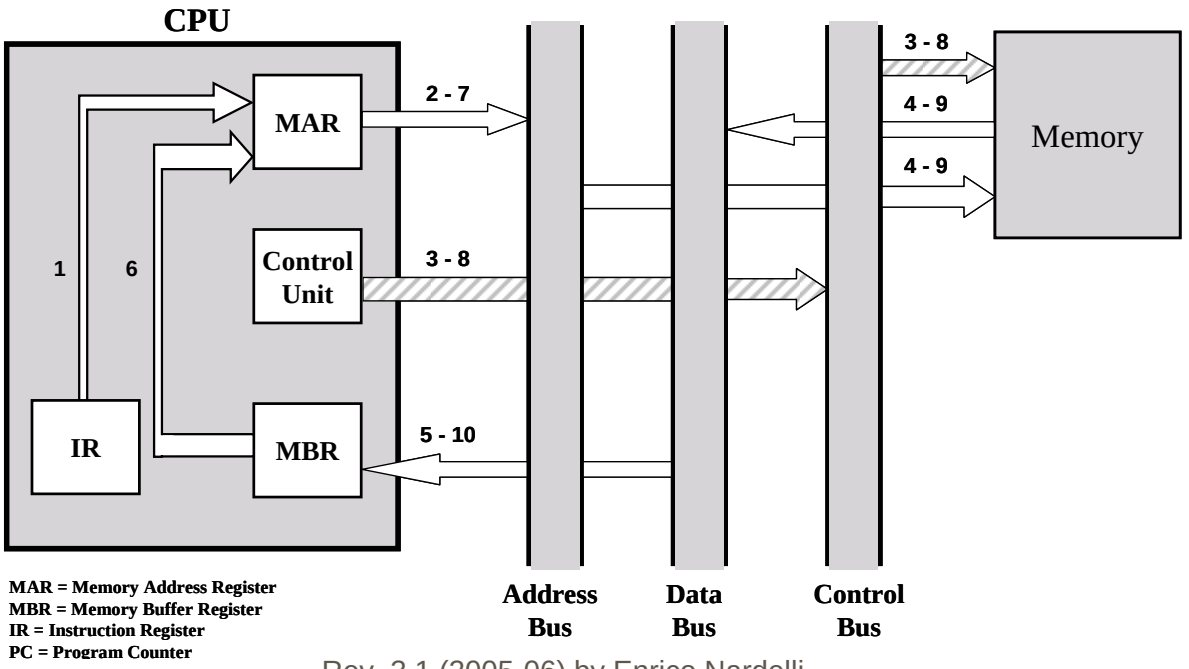
Sequence of micro-operations for register indirect addressing

- t_1: MAR <- ($IR_{register\text{-}address}$) <$DF_1$ DF_2 >
- t_2: MBR <- (memory) <DF_3 DF_4 DF_5 DF_6 >



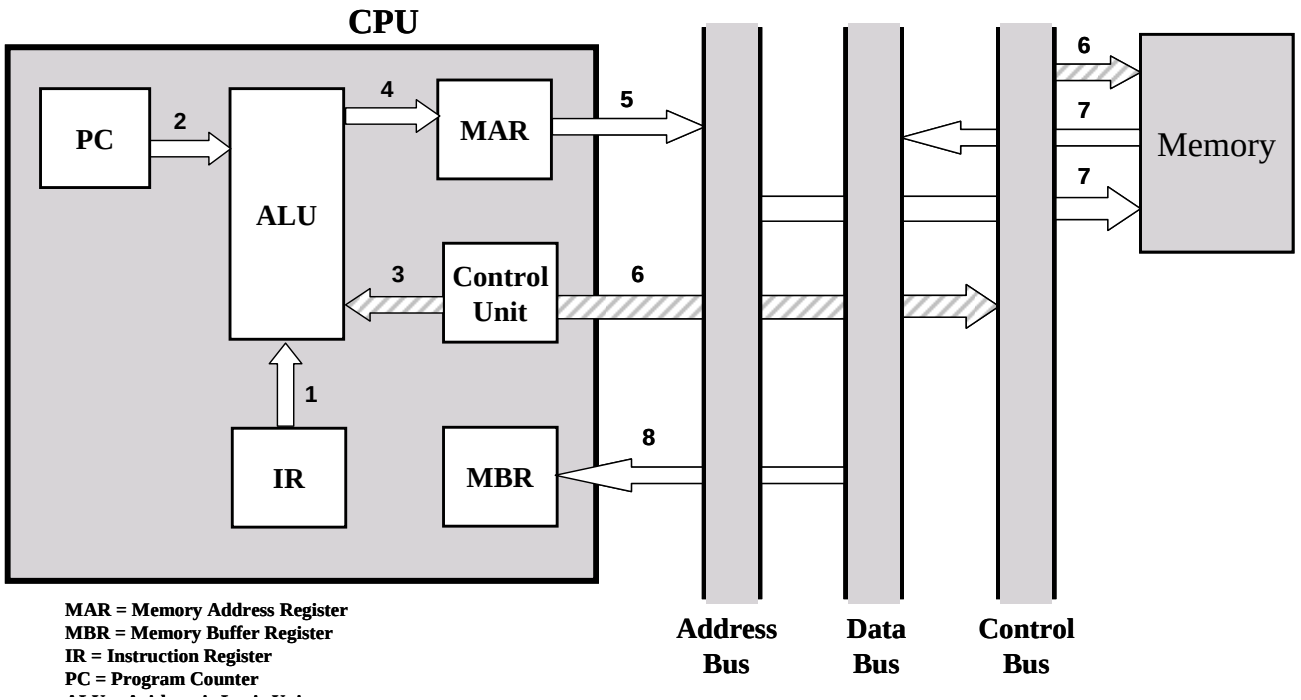
Sequence of micro-operations for indirect addressing

- t_1: MAR <- ($IR_{address}$) <DF_1 >
- t_2: MBR <- (memory) <DF_2 DF_3 DF_4 DF_5 >
- t_3: MAR <- (MBR) <DF_6 >
- t_4: MBR <- (memory) <DF_7 DF_8 DF_9 DF_{10} >



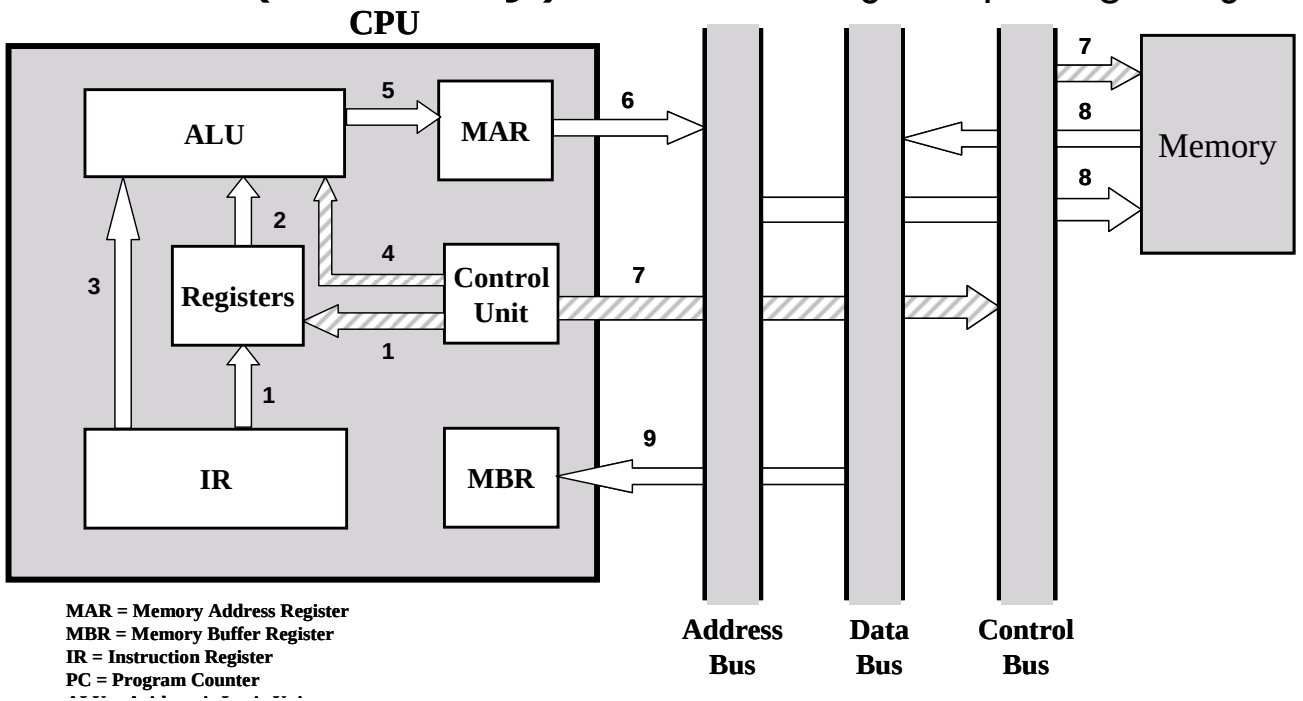
Sequence of micro-operations for relative addressing

- t_1: MAR <- (IR) + (PC) $<DF_1\ DF_2\ DF_3\ DF_4>$
- t_2: MBR <- (memory) $<DF_5\ DF_6\ DF_7\ DF_8>$



Sequence of micro-operations for base and indexed addressing

- t_1: MAR <- (($IR_{register\text{-}address}$)) + ($IR_{address}$)
  $<DF_1\ DF_2\ DF_3\ DF_4\ DF_5>$
- t_2: MBR <- (memory) $<DF_6\ DF_7\ DF_8\ DF_9>$

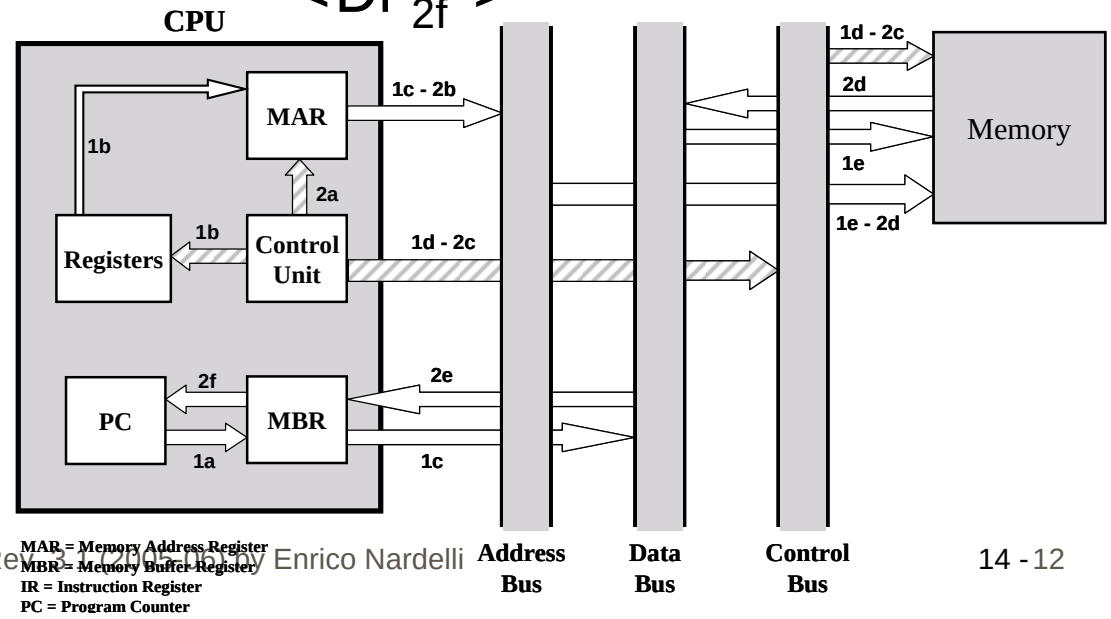


Sequence of micro-operations for combination of displacement and indirect addressing

- Try them yourself !

Sequence of micro-operations for interrupt handling

- t_1: MBR <- (PC) $<DF_{1a}>$
  MAR <- (Stack-Pointer) $<DF_{1b}>$
- t_2: Memory <- (MBR) $<DF_{1c}\ DF_{1d}\ DF_{1e}>$
- t_3: MAR <- Interrupt_Code $<DF_{2a}>$
- t_4: MBR <- (Memory) $<DF_{2b}\ DF_{2c}\ DF_{2d}\ DF_{2e}>$
- t_5: PC <- (MBR) $<DF_{2f}>$



Micro-operation sequencing for the execution phase (1)

- Different for each instruction

- MUL R1 X - add the contents of location X to Register 1 and store result in Register 1 and 2

  Assuming that content of cell at address X is in MBR after the operand fetch phase:

  - t_1: ALU <- (R1) * (MBR)
  - t_2: R1 <- $(ALU)_{low}$

    R2 <- $(ALU)_{high}$

Micro-operation sequencing for the execution phase (2)

- ISZ X - increment memory cell X and skip if it's zero

  Assuming that content of cell at address X is in MBR after the operand fetch phase:

  - t_1: MBR <- (MBR) + 1
  - t_2: memory <- (MBR)

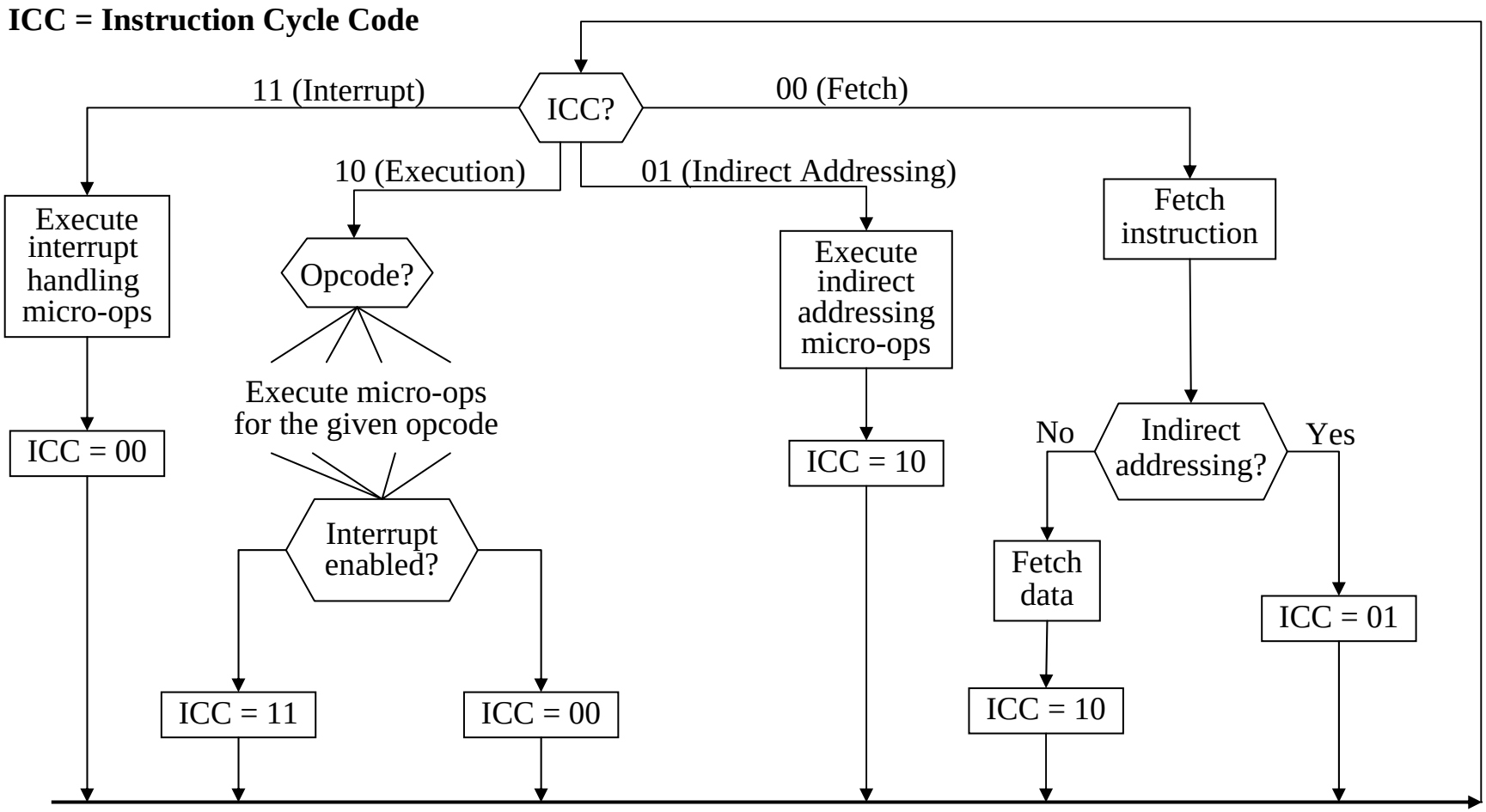
    IF (MBR) == 0 THEN PC <- (PC) + 1

- Note:
  - IF-THEN is a single micro-operation

Micro-operation sequencing for the execution phase (3)

- PPJ X - Save in stack the return address and jump to address X
  - t_1: MBR <- (PC)
    MAR <- (SP)
  - t_2: memory <- (MBR)
    SP <- (SP) + 1
    PC <- ($IR_{address}$)

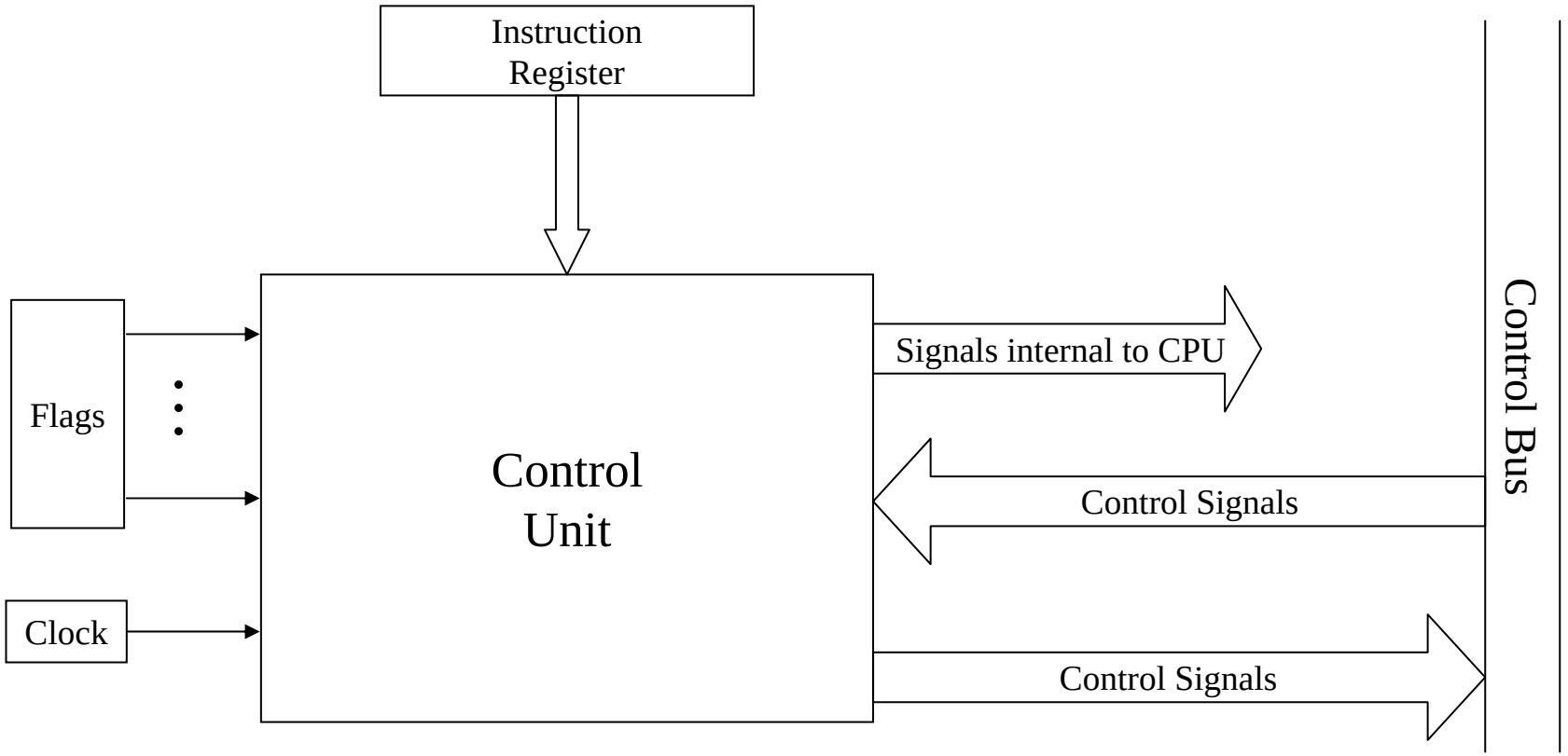
A simplified flow diagram for the execution of instruction cycle



Functions of Control Unit

- Sequencing
  - Causing the CPU to step through a series of micro-operations
- Execution
  - Causing the execution of each micro-op
- ALL THESE ACTIONS are performed by means of **Control Signals**

A simplified data flow diagram of a Control Unit



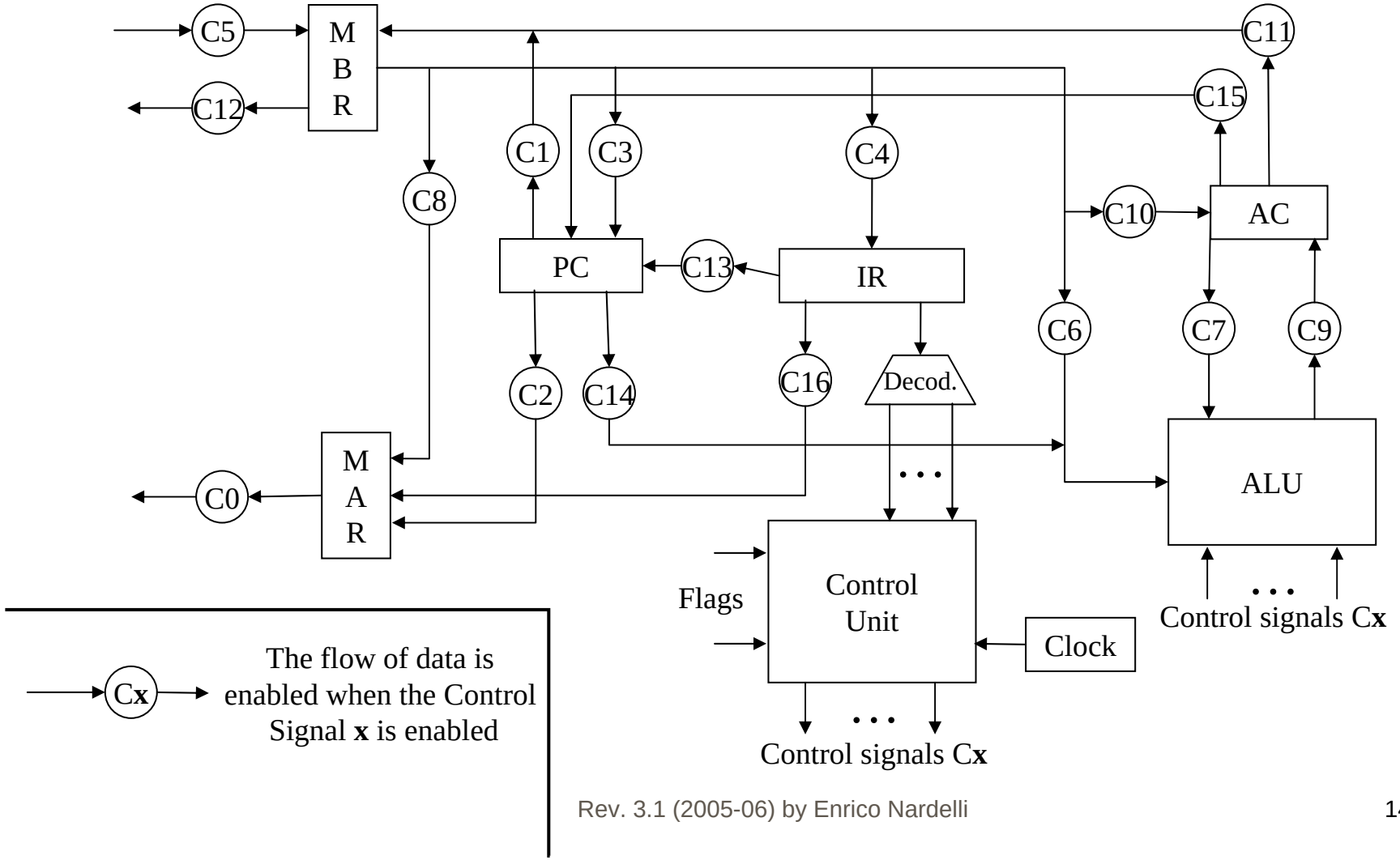
Input Signals

- Clock
  - One micro-op (or set of parallel ops) per clock cycle
  - Different signals are needed for different steps
- Instruction register
  - Op-code for current instruction
  - Determines which micro-instructions are performed
- Flags
  - State of CPU
  - Results of previous operations
- Control Bus
  - Interrupts
  - Acknowledgments

Output Signals

- To other CPU components
  - For data movement
  - To activate specific functions
- To the Control Bus
  - To control memory
  - To control I/O modules
- Output signals from the control unit **make all micro-operations happen**

Example of Control Signal use in a simplified schema of a CPU



Example of Control Signal Sequence – Instruction Fetch (1)

- t_1: MAR <- (PC)
  - Control unit (CU) activates signal C2 to open gate between PC and MAR
- t_2: MBR <- (memory)
  - CU activates C0 to open gate between MAR and address bus
  - CU activates the memory read control signal (CR - not shown) to the memory
  - CU activates C5 to open gate between data bus and MBR

Example of Control Signal Sequence – Instruction Fetch (2)

- t_3: PC <- (PC) +1
  - In the simple schema shown there is no direct data path from ALU to PC hence the micro-op has to be split in two (ALU has internally an output buffer to store result):
  - t_{3-1}: ALU <- (PC) CU activates C14
  - increment ALU CU act. control signal CA (not
  - shown) for ALU
  - t_{3-2}: AC <- (ALU) CU activates C9
  - PC <- (AC) CU activates C15
- t_4: IR <- (MBR)
  - CU activates C4

Example of Control Signal Sequence – Instruction Fetch (3)

- Optimization
  - t_2 and $t_{3\text{-}1}$ can be executed together
  - $t_{3\text{-}2}$ and t_4 can be executer together
- New organization
  - t_1: MAR <- (PC) C2
  - t_2: MBR <- (memory) C0 CR C5
    (PC)+1 in ALU C14 CA
  - t_3: AC <- (ALU) C9
    PC <- (AC) C15
    IR <- (MBR) C4

Example of Control Signal Sequence - Indirect Addressing

- t_1: MAR <- ($IR_{address}$)
  - CU activates C16 to open gate between IR and MAR
- t_2: MBR <- (memory)
  - CU act. C0 to open gate between MAR and address bus
  - CU act. the memory read control signal (CR)
  - CU act. C5 to open gate between data bus and MBR
- t_3: MAR <- (MBR)
  - CU activates C8 to open gate between MBR and MAR
- t_4: MBR <- (memory)
  - CU activates C0, CR and C5 as above

Other Examples of Control Signal Sequence

- Direct Addressing
- Relative Addressing

- Try them yourself !

Limitations

- The simplified schema of a CPU's does not show registers, hence we cannot show
  - Register addressing
  - Register indirect addressing
  - Base addressing
  - Indexed addressing
  - Combination of displacement and indirect addressing

  - Try adding to the simplified schema one or more of the above addressing modalities and derive the required micro-operations!

Internal Organization of CPU

- Usually a single internal bus
  - less complex then having direct data paths between registers and ALU
- Gates control movement of data onto and off the internal bus
- Control signals control also data transfer to and from external systems bus
- Temporary registers in input to ALU are now needed for proper operation of ALU

Hardwired Implementation (1)

- Consider the control unit as a combinational circuit
  - Outputs of the circuit are the control signals
  - Inputs of the circuit are status signal for ICC and opcode bits
  - For each configuration of inputs produce a proper output
  - That is, the activation of a given control signal Cn has to happen when this condition is true OR this condition is true OR ...

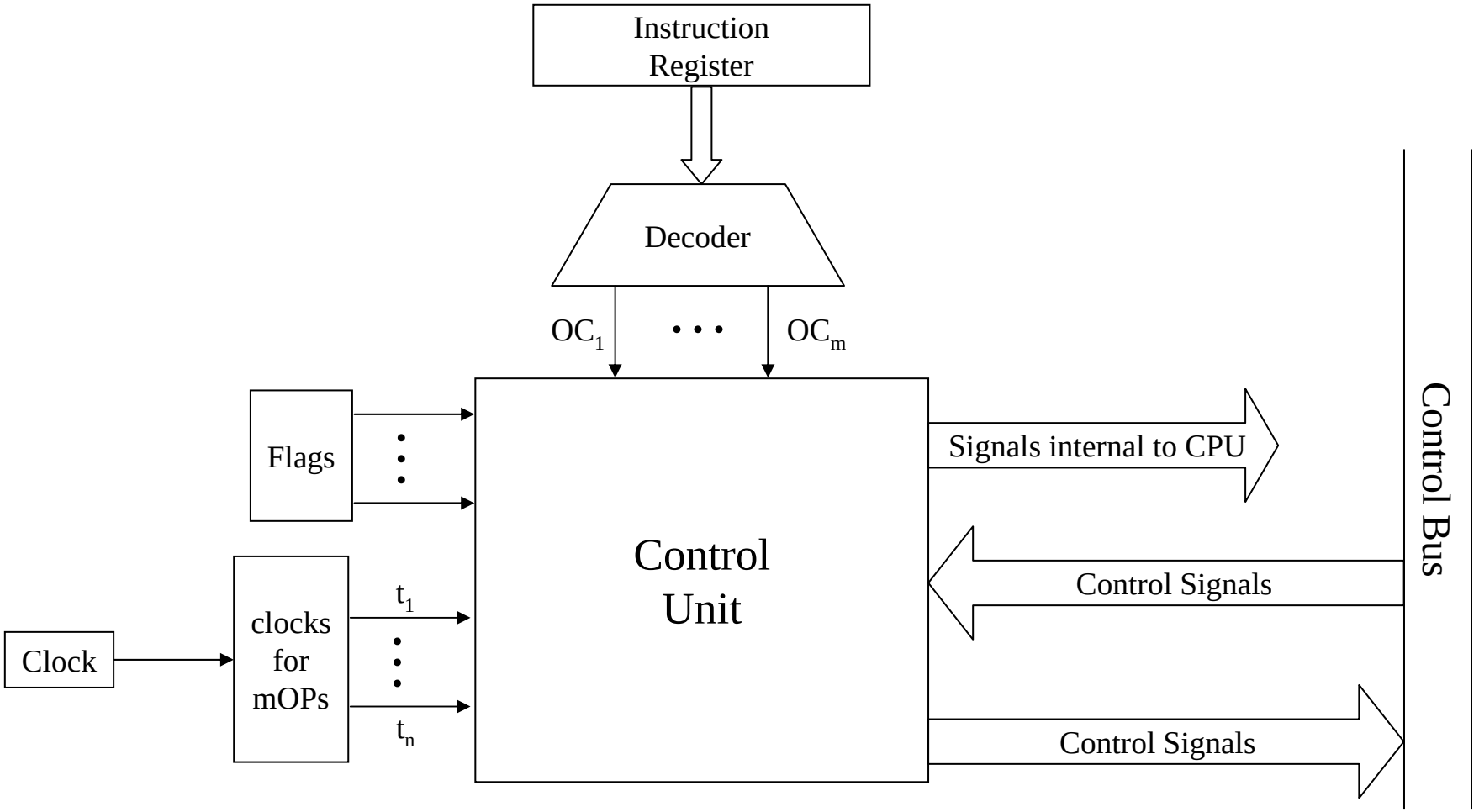
Hardwired Implementation (2)

- Example:
  - Control signals P and Q code ICC: then fetch is coded by P'Q', indirect by P'Q, execute by PQ', and interrupt by PQ
  - Opcode bits are further control signals
  - Boolean expression activating C5 in the simplified schema of a CPU:

    $C5 = P'Q't_2 + P'Q(t_2 + t_4) + PQ'B$

    - where *B* is the boolean expression representing, for all opcodes activating C5, all micro-operations actually activating it
    - E.g.: if C5 is activated by opcode 3 during t_2 and t_4 and by opcode 7 during t_3 and t_4 then *B* is: $OC3(t_2+t_4)+OC7(t_3+t_4)$

A simplified data flow diagram of a Control Unit for the hardwired implementation



Problems with the Hardwired Implementation

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instruction