COMPUTER ARCHITECTURE

Code: CT173

Part IV: John Von Neumann Architecture

MSc. NGUYEN Huu Van LONG

Department of Computer Networking and Communication,

College of Information & Communication Technology,

CanTho University

Agenda

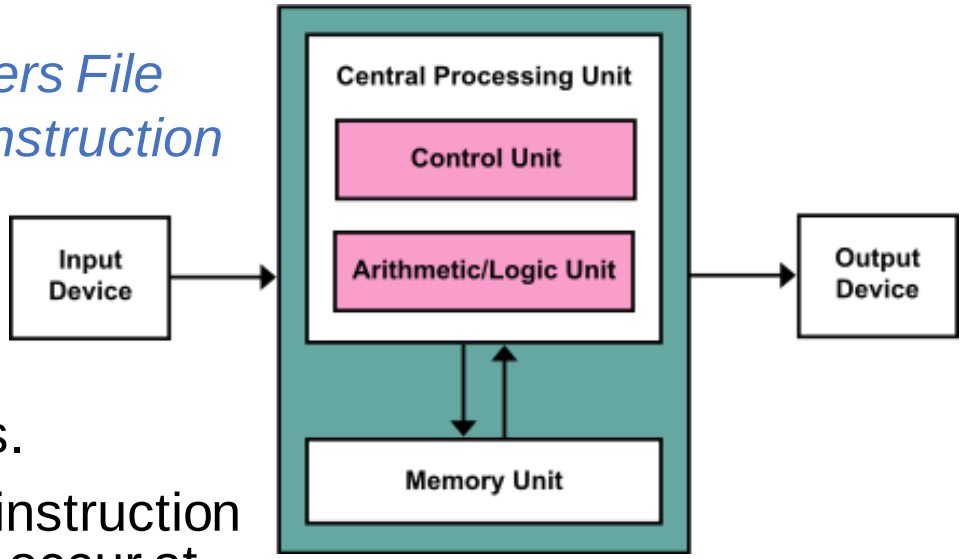
- John Von Neumann bibliography
- **John Von Neumann architecture concept**
- Control Path
  - Hardware Control
  - Firmware Control
- Data Path
  - Single Cycle Data path
  - Multi Cycle Data path
- **Phase of execution (based on MIPS)**

Bibliography of John Von Neumann

- December 28, 1903 - February 8, 1957
- Hungarian-American mathematician, physicist, computer scientist and polymath
- The foremost mathematician of his time and said to be “the last representative of the great mathematicians”
- Contribution in many fields:
  - ***Mathematics:*** foundation of mathematics, functional analysis, ergodic theory, representation theory, operator algebras, geometry, topology and numerical analysis
  - ***Physics***: quantum mechanics, hydrodynamics, quantum statistical mechanics
  - ***Economics***: game theory
  - ***Computing***: Von Neumann architecture, linear programming, self replicating machines, stochastic computing
  - ***Statistics***
- He wrote the ***First Draft of a Report on the EDVAC*** (an incomplete 101-page document). It contains the first published description of the logical design of a computer using the stored-program concept, *which has controversially come to be known as the Von Neumann Architecture*

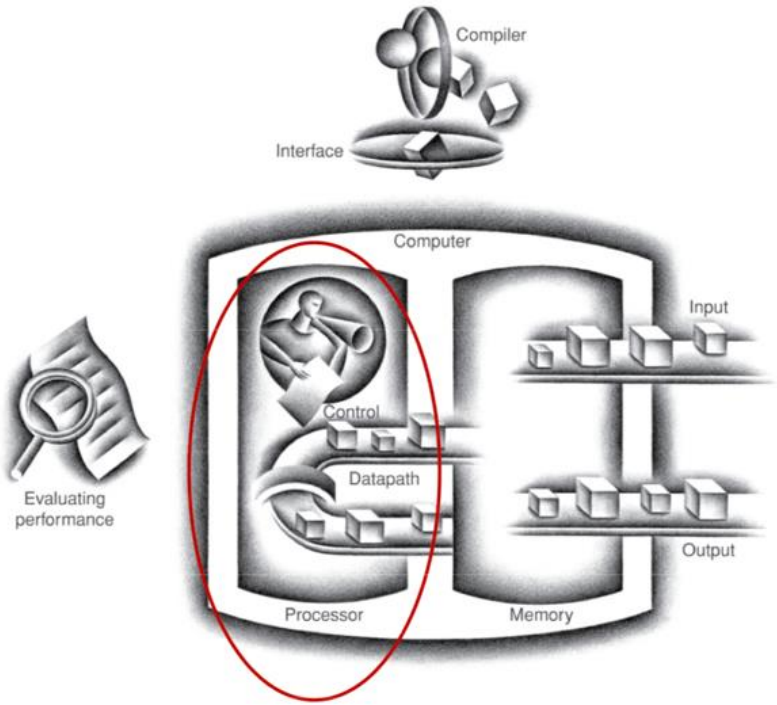
John Von Neumann Architecture

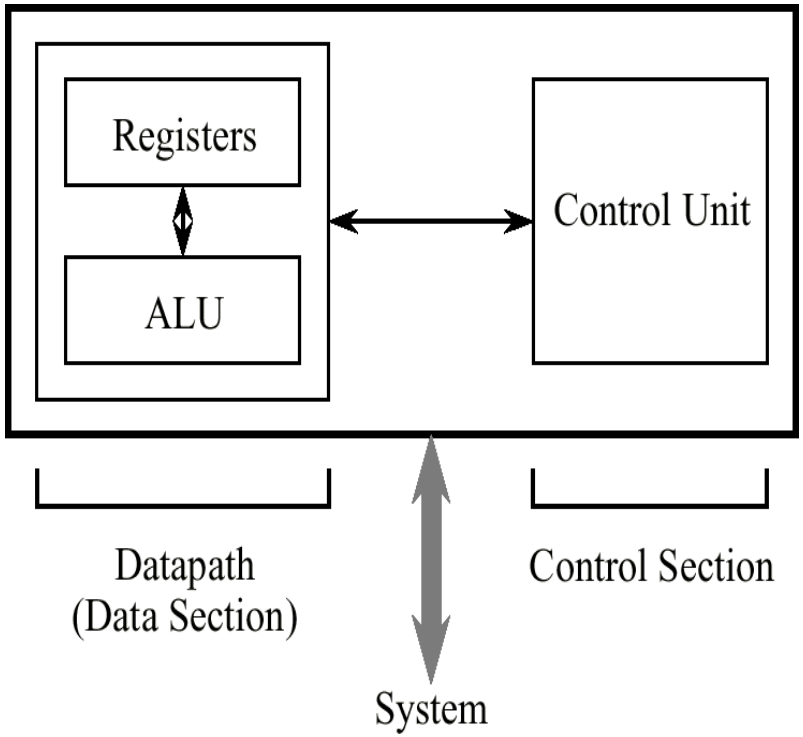
- The **Von Neumann architecture**, which is also known as **Princeton architecture**
- This architecture describes a design for an electronic digital computer with parts consisting of:
  - A *Processing Unit* containing an *Arithmetic Unit* and *Registers File*
  - A *Control Unit* containing an *Instruction Register* and `
  - A *Memory* to store both data and instruction
  - External *Mass Storage*
  - *Input and Output* mechanisms.
- In Von Neumann architecture: an instruction fetch and a data operation cannot occur at the same time *because they share a common bus* → **Bottleneck**: limits the performance of the system → The modern architecture of computer (parallel architecture) should due with this challenge



Datapath and Controlpath

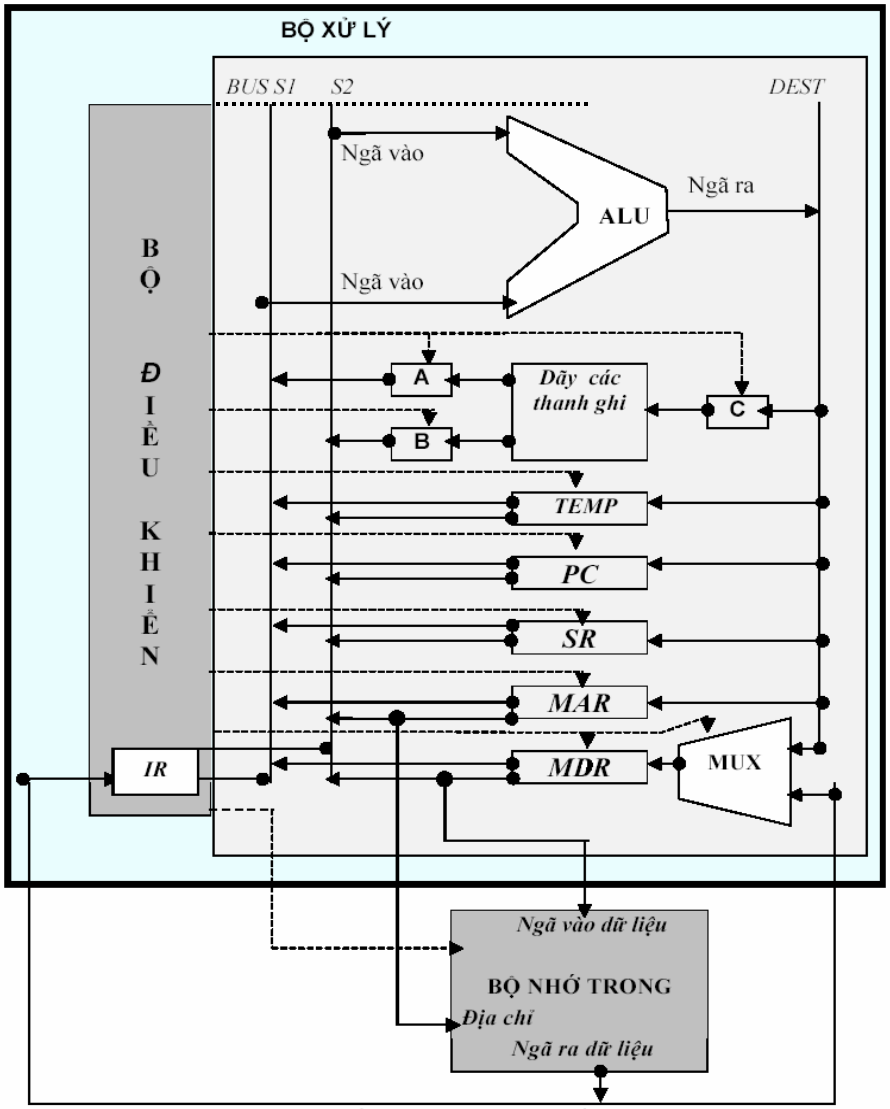
- **Datapath**: components of the processor that ***perform*** *arithmetic/logic operations* or *processing data from/to memory*
- **Controlpath**: components of the processor that ***commands*** *datapath, memory, I/O devices* according to the instructions of the memory





Controlpath

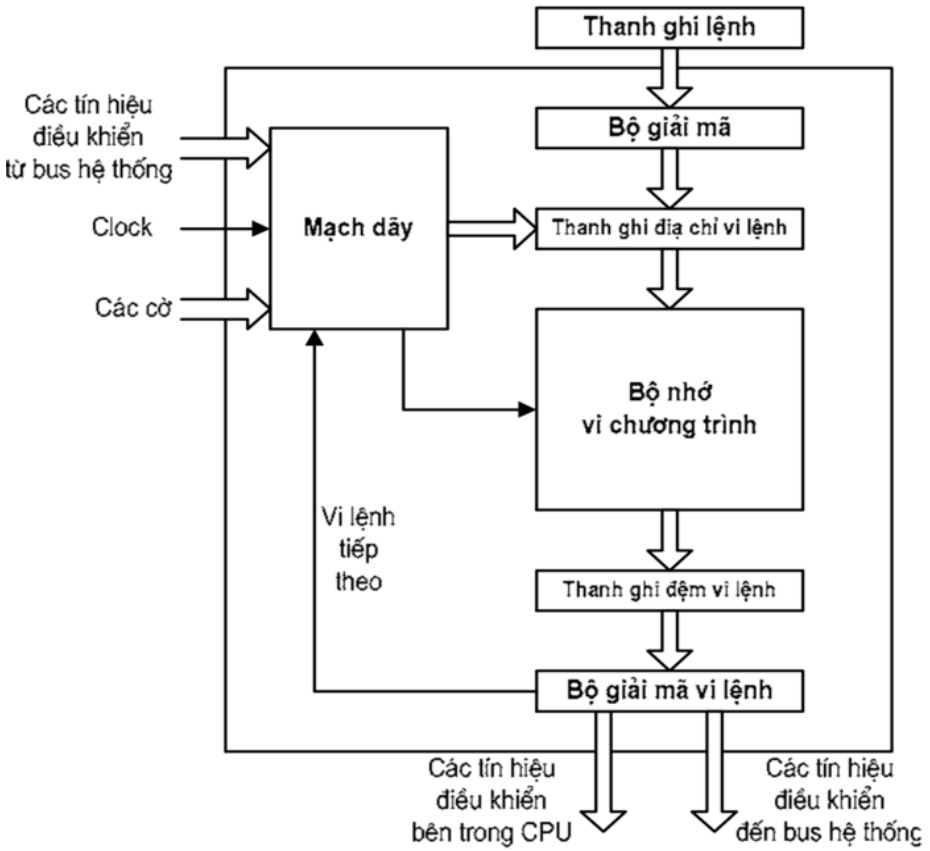
- Controlpath has the responsibility to *create signal* in cases:
  - *Move instruction or a block of instructions* to Instruction Register – IR in CPU
  - *Increase the value of PC register by d* (with d is the size of Word) to point to the next instructions need to be executed.
  - *Decode the loaded instruction(s)* in Instruction Register(s) to process the task.
  - *Move data* from memory into operands (registers)
  - *Execute the instruction and store the result* into memory
- Two polarizing different microarchitecture approaches: **Microprogrammed control units** and **Hardwired control units**



Hình III.1: Tổ chức của một xử lý điển hình
(Các đường không liên tục là các đường điều khiển)

Microprogrammed Control Unit

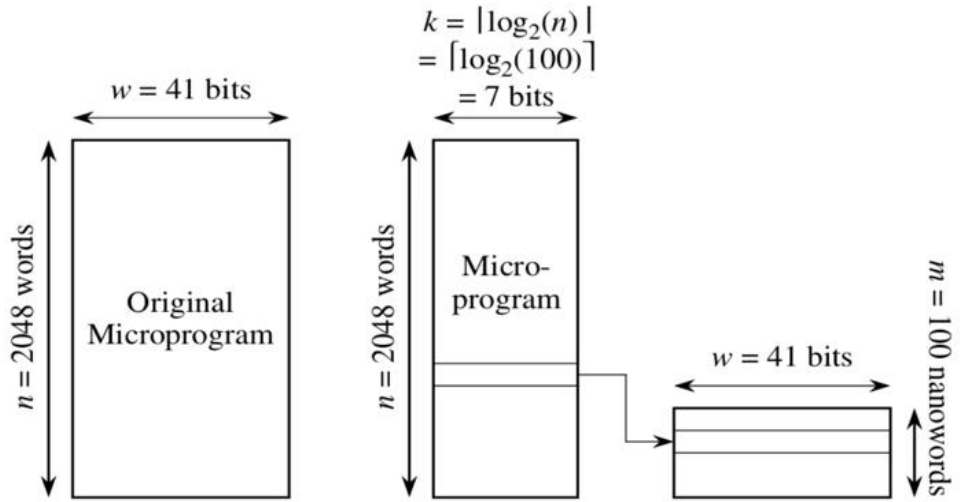
- To process an instruction defined in ISA, CPU *uses one or several microprograms* that are stored in **Read Only Memory – ROM** under binary object code form
- A *microprogram* contains a lots of sequential *microinstructions* that is represented by an invisible assembly language
- The *microprogram* is often referred to as **firmware** because it bridges the gap between the hardware and the software.
- The main purpose of the firmware is to interpret a user-visible instruction set.
- **Using popularly in CISC architecture**.



Microprogrammed Control Unit

- The firmware approach is *slow*, and *consumes a huge amount of hardware* in comparison with the hardwired approach
- However, the firmware approach is *flexible* and *simplifies the process of modifying the instruction set*
- The significant hardware consumed could be reduced to a degree through using nanoprogram but the *delay in execution of microinstruction will be increased*

Address	Operation Statements	Comment
0:	R[ir] ← AND(R[pc],R[pc]); READ;	/ Read an ARC instruction from main memory
1:	DECODE;	/ 256-way jump according to opcode
	/ **sethi**	
1152:	R[rd] ← LSHIFT10(ir); GOTO 2047;	/ Copy imm22 field to target register
	/ **call**	
1280:	R[15] ← AND(R[pc],R[pc]);	/ Save %pc in %r15
1281:	R[temp0] ← ADD(R[ir],R[ir]);	/ Shift disp30 field left
1282:	R[temp0] ← ADD(R[temp0],R[temp0]);	/ Shift again
1283:	R[pc] ← ADD(R[pc],R[temp0]);	/ Jump to subroutine
	GOTO 0;	



Total Area = $n \times w$ = $2048 \times 41 = 83{,}968$ bits

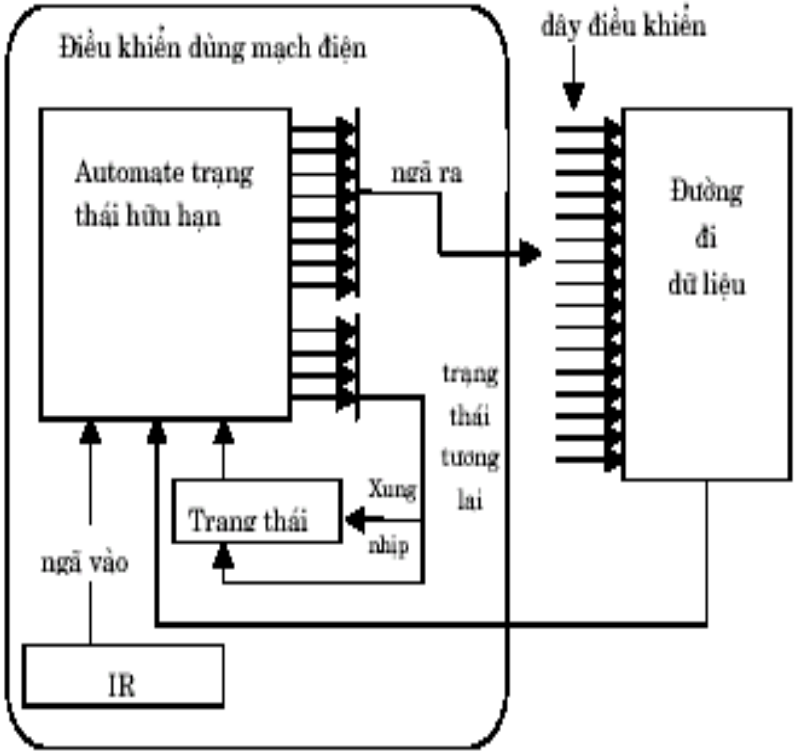
Microprogram Area = $n \times k = 2048 \times 7$ = 14,336 bits

Nanoprogram Area = $m \times w = 100 \times 41$ = 4100 bits

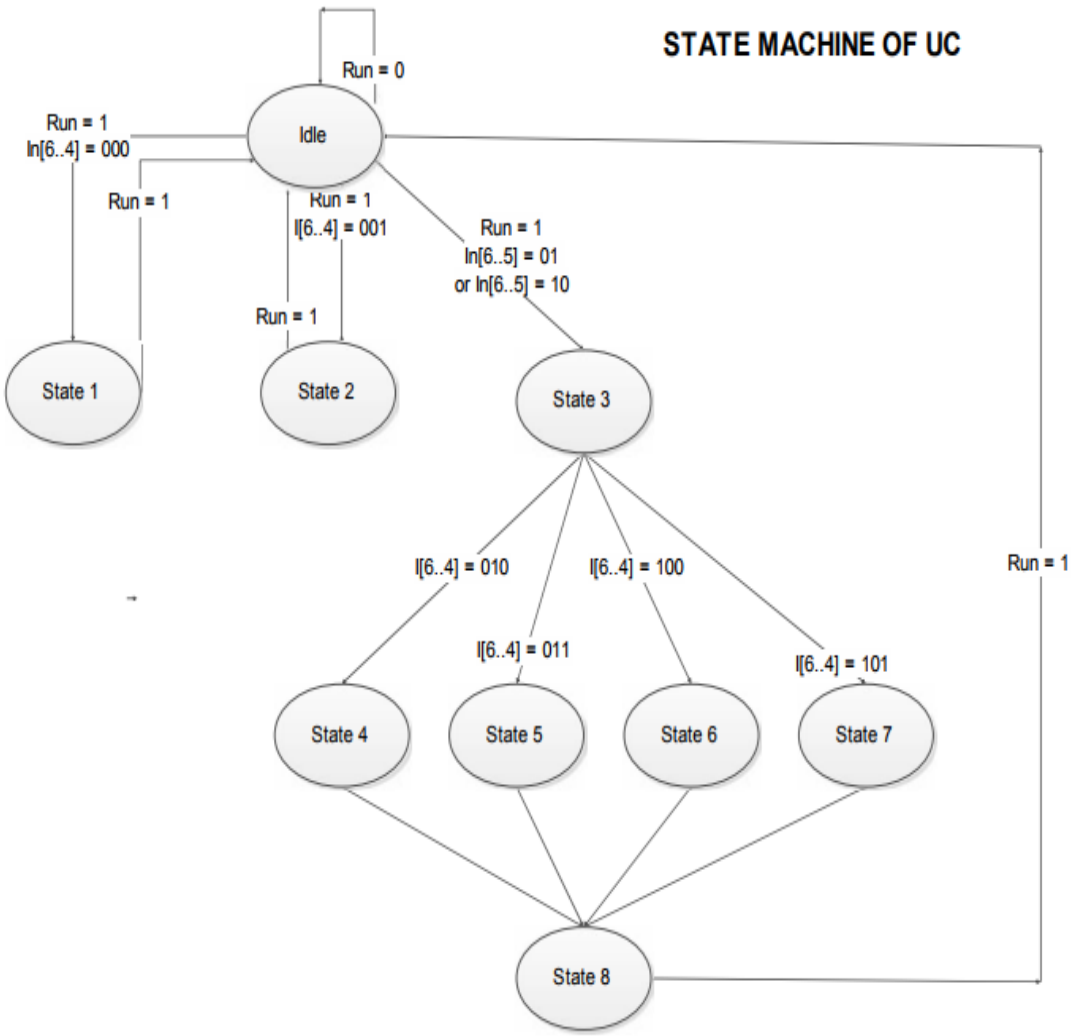
Total Area = 14,336 + 4100 = 18,436 bits

Hardware Control Unit

- Hardware control unit is directly implemented with flip-flops and logic gates
  - *Flip-flops maintain state information*
  - *Logic gates implements transitions among the states*
- A **finite state machine – FSM** in hardwired implementation replaces sequential steps in microprogram
- The hardwired approach is *fast*, and consumes a *small amount of hardware* but *difficult to modify*, and typically results in *more complicated implementations*
- **Hardwired Description Language – HDL** is frequently used to represent the control structure, i.e. **VHDL**
- **Using popularly in RISC architecture**



Hardware Control Unit



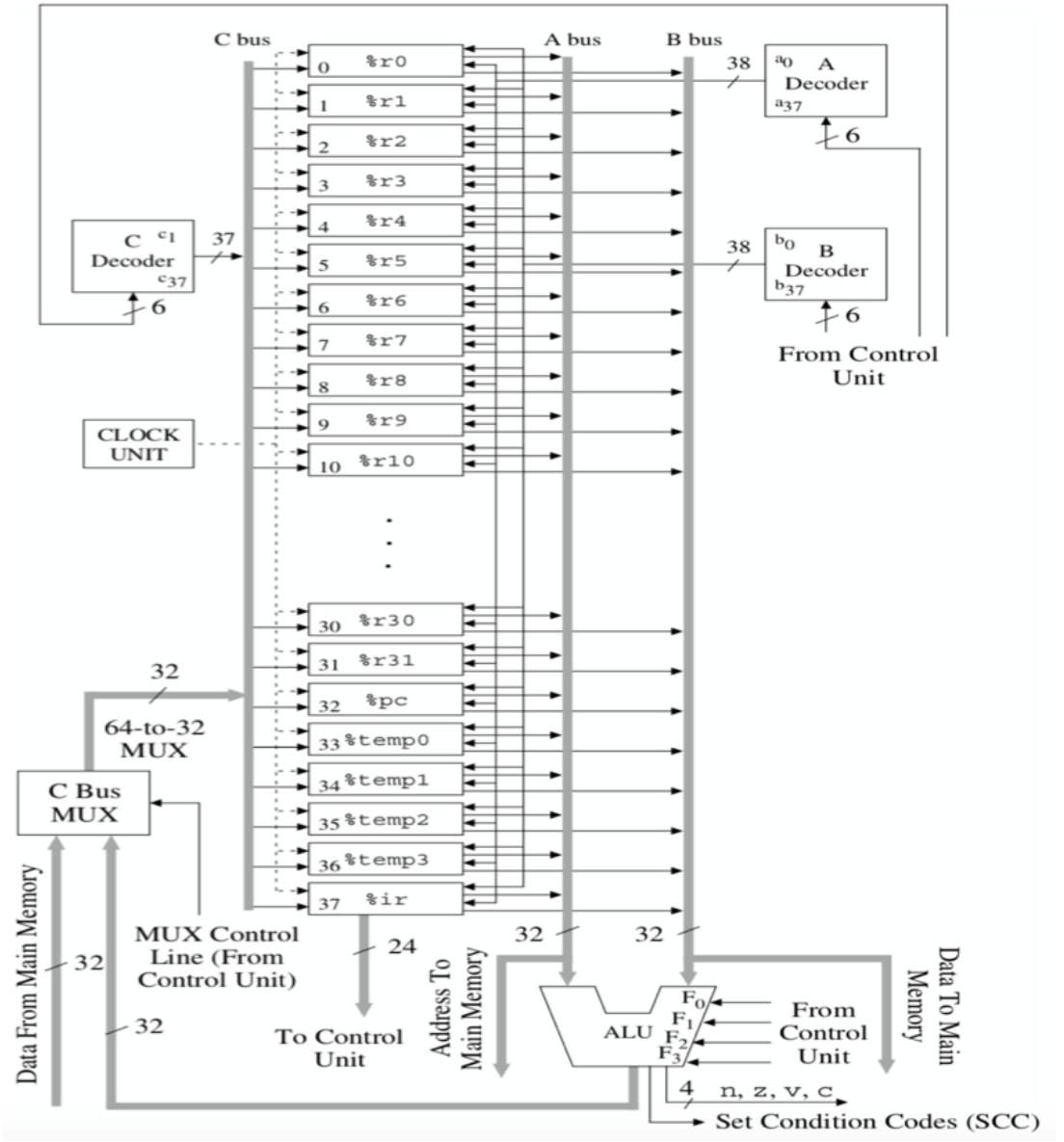

```
--Declare the structure of entity Mux Full Component
LIBRARY IEEE ;
ARCHITECTURE Behavioral OF ControlUnit IS
  type state_type is (s0,s1,s2,s3,s4,s5,s6,s7,s8);
  signal change_state: state_type;
  BEGIN
    --Process for FSM
    process (inputClock)
    begin
      if (inputReset='1') then
        change_state <= s0;  --default state on reset.
      elsif (inputClock'event and inputClock = '1' ) then

        if (inputRun = '1') then
          case input64 is
            when "000" => change_state <= s1;
            when "001" => change_state <= s2;
            when "010" => change_state <= s3;
            when "011" => change_state <= s3;
            when "100" => change_state <= s3;
            when "101" => change_state <= s3;
            when others => --Do nothing
          end case;
        end if;

        if (change_state = s3 and inputRun = '1') then
          case input64 is
            when "010" => change_state <= s4;
            when "011" => change_state <= s5;
            when "100" => change_state <= s6;
            when "101" => change_state <= s7;
            when others => --Do nothing
          end case;
        end if;
```

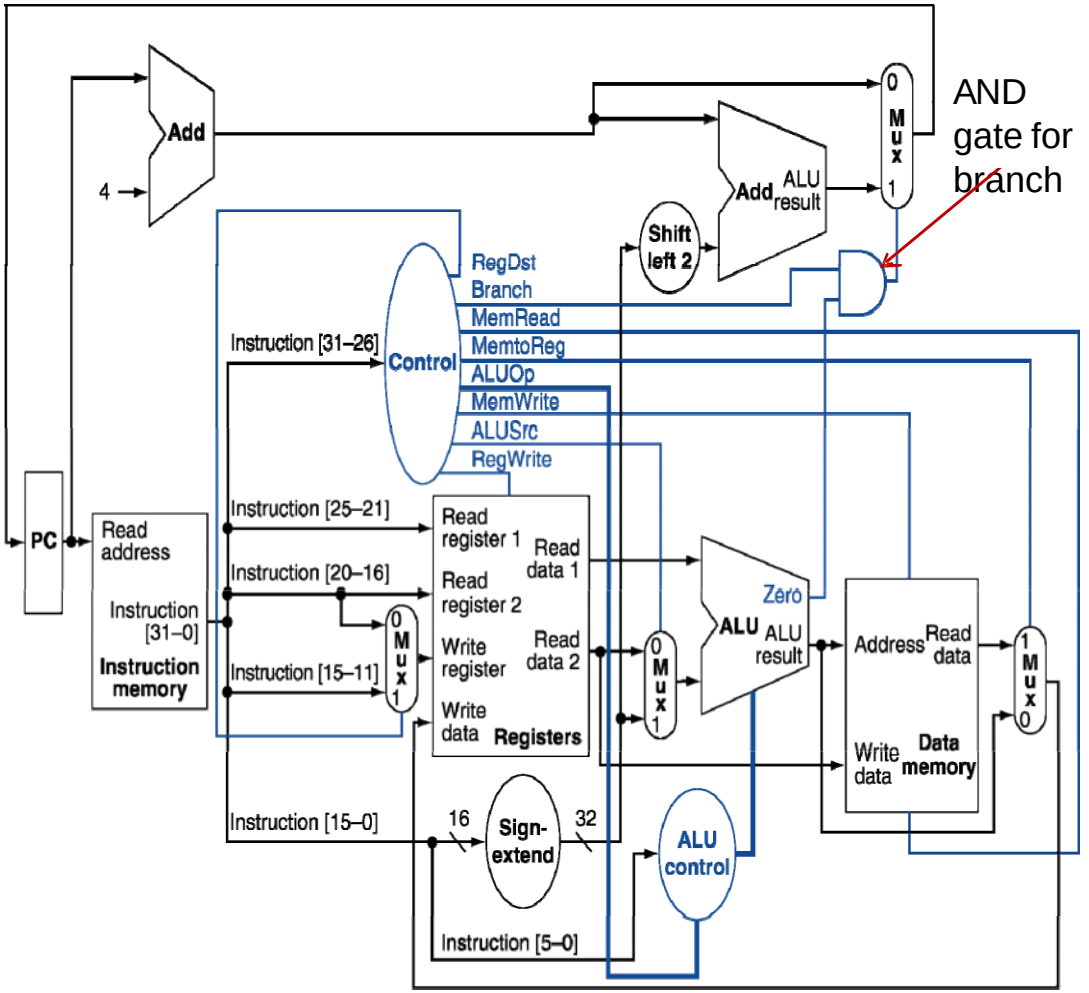
Datapath Elements

- **Datapath elements**
  - Registers file
  - ALUs
  - Sign extension
  - Instruction memory
  - Data memory
  - Memory
  - Multiplexers
  - Shifter...



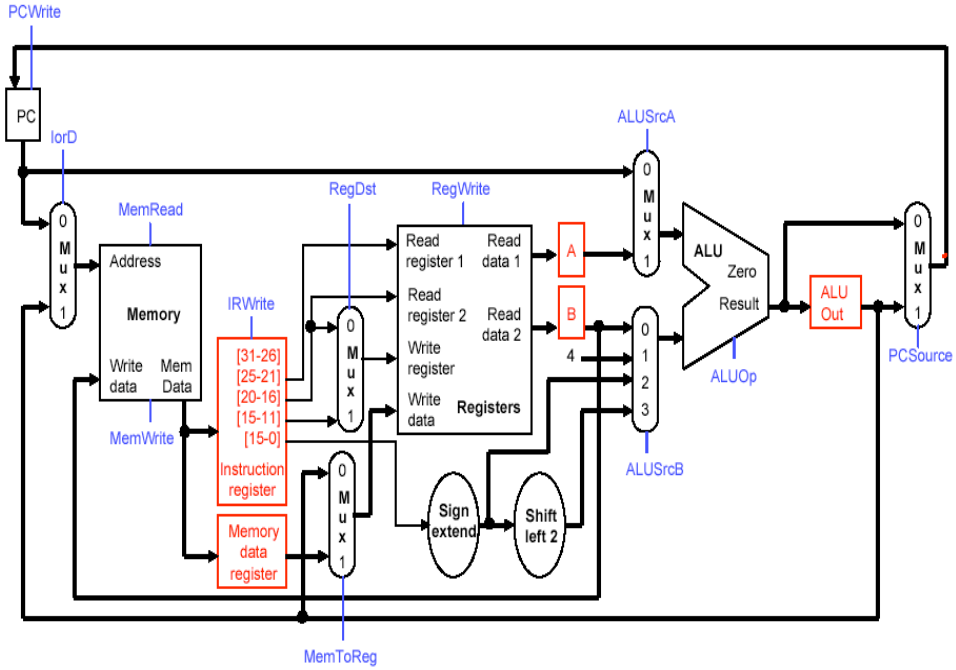
Single Cycle Datapath (i.e. MIPS)

- Building a typical **Datapath** needs to consider:
  - The Datapath portion to *fetch an instruction into CPU*
  - The Datapath portion for *arithmetic/logic operation instructions*
  - The Datapath portion for *Load/Store instructions*
  - The Datapath portion for *Branch instructions*
- *All instructions have same clock cycle time length (T)* which is determined by the longest path
  - *CPI is always 1*
  - Wasted time in case the instruction is short



Multi Cycle Datapath (i.e. MIPS)

- The instruction could be divided into arbitrary number of steps, i.e. in MIPS64, there are 5 steps (stages)
- Clock Cycle Time (T) is short → CPI is various according the different type of instructions → faster in average execution time compared to Single Cycle Datapath
- **Intermediate registers are essential**.
- Unique Memory (Princeton architecture) could be replaced by Data Memory and Instruction Memory



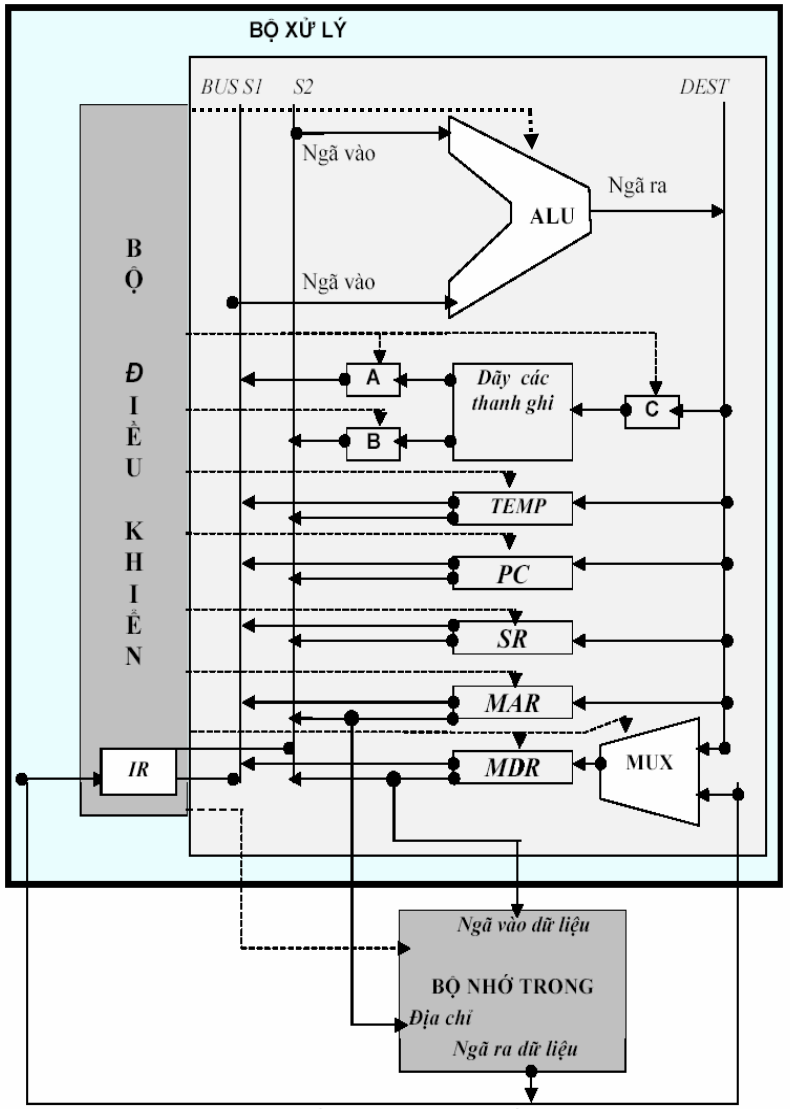
Dedicated Registers	Name	Description
Program Counter Register	**PC**	Indicates the next instruction (by physical address) to compile of a program which stored in main memory.
Instruction Register	**IR**	Indicates the current instruction (by physical address) in processing inside the CPU
Stack Pointer Register	**SP**	Indicates the top of Stack architecture (by physical address)
Base Register	**BR**	Indicates the starting point of the program inside main memory (by physical address)
Memory Address Register	**MAR**	Indicates the memory address need to be interacted with Datapath section in CPU
Memory Data Register	**MDR**	Indicates the temporary value in the middle of interaction between Datapath section in CPU and a memory address indicated by MAR

Single Cycle vs Multi Cycle

	Single Cycle	Multiple Cycle
Instructions Subdivided?	No	Yes, into arbitrary number of steps
Clock Cycle Time	Long (long enough for the slowest instruction)	Short (long enough for the slowest instruction step)
CPI	1 clock cycle per instruction (by definition)	Variable number of clock cycles per instruction
Number of Instructions Executing at the Same Time	1	1
Control Unit	Generates signals for entire instruction	Generates signals for instruction's current step, and keeps track of the current step
Duplicate Hardware?	Yes, since we can use a functional unit (FU) for at most one subtask per instruction	No, since the instruction generally is broken into single-FU steps
Extra Registers?	No	Yes, to hold the results of one step for use in the next step
Performance	Baseline	Slightly slower to moderately faster than single cycle, the latter when the instructions steps are well balanced and a significant fraction of the instructions take less than the maximum number of clock cycles

Phase of execution (MIPS)

- MIPS could be implemented with single cycle datapath or multiple cycle datapath.
- In case multiple cycle datapath implementation, every MIPS instruction has at most 5 clock cycles
- ***Stage 1 - Instruction Fetch (IF)***
  - Send the content of Program Counter register to Instruction Memory and fetch the current instruction from Instruction Memory .
  - Update the PC to the next sequential address by adding 4 to the PC (since each instruction is 4 bytes).
- ***Stage 2 - Instruction Decode and Register Read (ID)***
  - Decode the current instruction (fixed-field decoding) and read from the Register File – RF of one or two registers corresponding to the registers specified in the instruction fields.
  - Sign-extension of the offset field of the instruction in case it is needed.



Hình III.1: Tổ chức của một xử lý điển hình
(Các đường không liên tục là các đường điều khiển)

Phase of execution (MIPS)

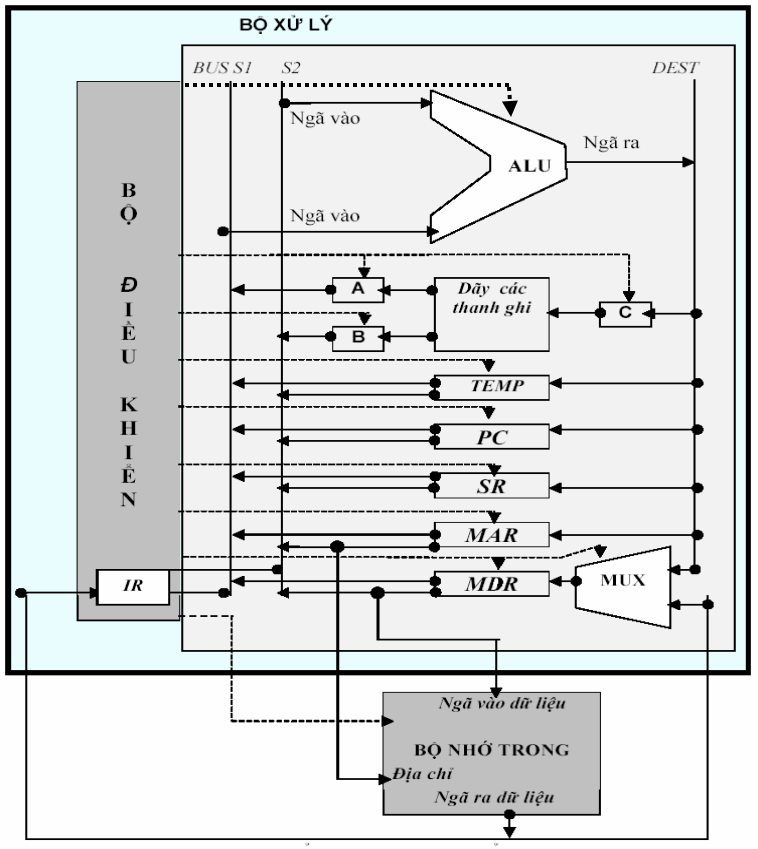
- ***Stage 3 - Execution (EX):*** The ALU operates on the operands prepared in the previous cycle depending on the instruction type
  - Register - Register ALU Instructions
    - ALU executes the specified operation on the operands read from the RF
  - Register - Immediate ALU Instructions
    - ALU executes the specified operation on the first operand read from the RF and the sign-extended immediate operand
  - Memory Reference
    - ALU adds the base register and the offset to calculate the effective address
  - Conditional branches
    - Compare the two registers read from RF and compute the possible branch target address by adding the sign-extended offset to the incremented PC

Phase of execution (MIPS64)

- ***Stage 4 - Memory Access (ME)***
  - *Load* instructions require a read access to the Data Memory using the effective address
  - *Store* instructions require a write access to the Data Memory using the effective address to write the data from the source register read from the RF
  - Conditional branches can update the content of the PC with the branch target address, if the conditional test yielded true
- ***Stage 5 - Write-Back (WB)***
  - Load instructions write the data read form memory in the destination register of the RF
  - ALU instructions write the ALU results into the destination register of the RF

Phase of execution (MIPS64)

- **Stage 1 - IF**
  - IR := M[MAR]
  - PC := PC + 4
- **Stage 2 - DE**
  - A := RS1 ← IR[25:21]
  - B := RS2 ← IR[20:16]
  - Offset := SE ← IR[15:0] (in case needed, i.e. load, store instruction)
- **Stage 3 - EX**
  - Arithmetic/logic operations: ALUOut := A op B
  - Memory reference:
    - Load instr: MAR := ALUOut ← B + Offset
    - Store instr: MAR := B + Offset; MDR := ALUOut ← A
  - Branch: if A = B then PC := ALUOut ← PC + A + B
- **Stage 4 - MEM**
  - Load Ins: MDR := M[MAR]
  - Store Ins: M[MAR] := MDR
- **Stage 5 - WB**
  - Load Ins: RD := MDR
  - Arithmetic/logic operations: RD := ALUOut



Instruction	Stage required				
BEQ, JMP, BR	IF	DE	EX		
Arith/Logic	IF	DE	EX		WB
Load	IF	DE	EX	MEM	WB
Store	IF	DE	EX	MEM	

Example: Phase of Execution

- LOAD R1, R2(100)
- STORE R3, R4
- ADD R5, R6, R7(200)
- BR R8
- BNRZ R9, +5

Question?