Advance Computer Architecture and x86 ISA

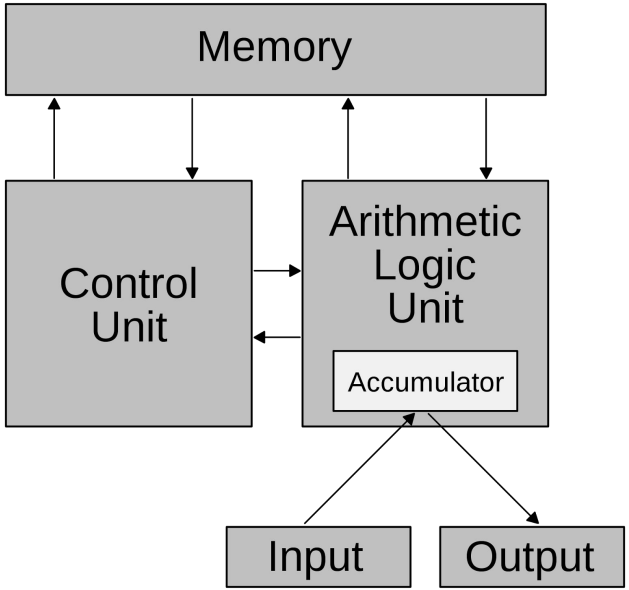
University of Science and Technology of Hanoi

MS. LE Nhu Chu Hiep

Von Neumann Processors

Remind ?

- Components:
  - Arithmetic Logic Unit
  - Control Unit
  - Memory
  - Input / Output
- Architecture:
  - **Accumulator based**



Instruction executions

- Instruction execution
  - Read instruction from memory
  - Decode instruction to execute
  - Execute the instruction
    - Control of the arithmetic logic unit to execute the instruction
  - Store the result
- Von neumann / Instruction cycle:
  - The time required by the CPU to execute one single instruction
  - In one clock cycle, the instruction could be executed in 4 stages
    - Fetch, decode, execute, and writeback

Instruction executions: read and decode

- Read the instruction from the memory
  - Access to the memory and load an instruction in the instruction register
  - The current address is stored in a specific register, called program counter (PC)
  - The program counter is incremented if no branch/jump is required, in the case of jump, the program counter is loaded with another value
- Decode instruction
  - After having read the instruction, the processor must know what to execute
  - Instruction decomposition
    - Operational code
    - Operands to use

Instruction executions: execute and store

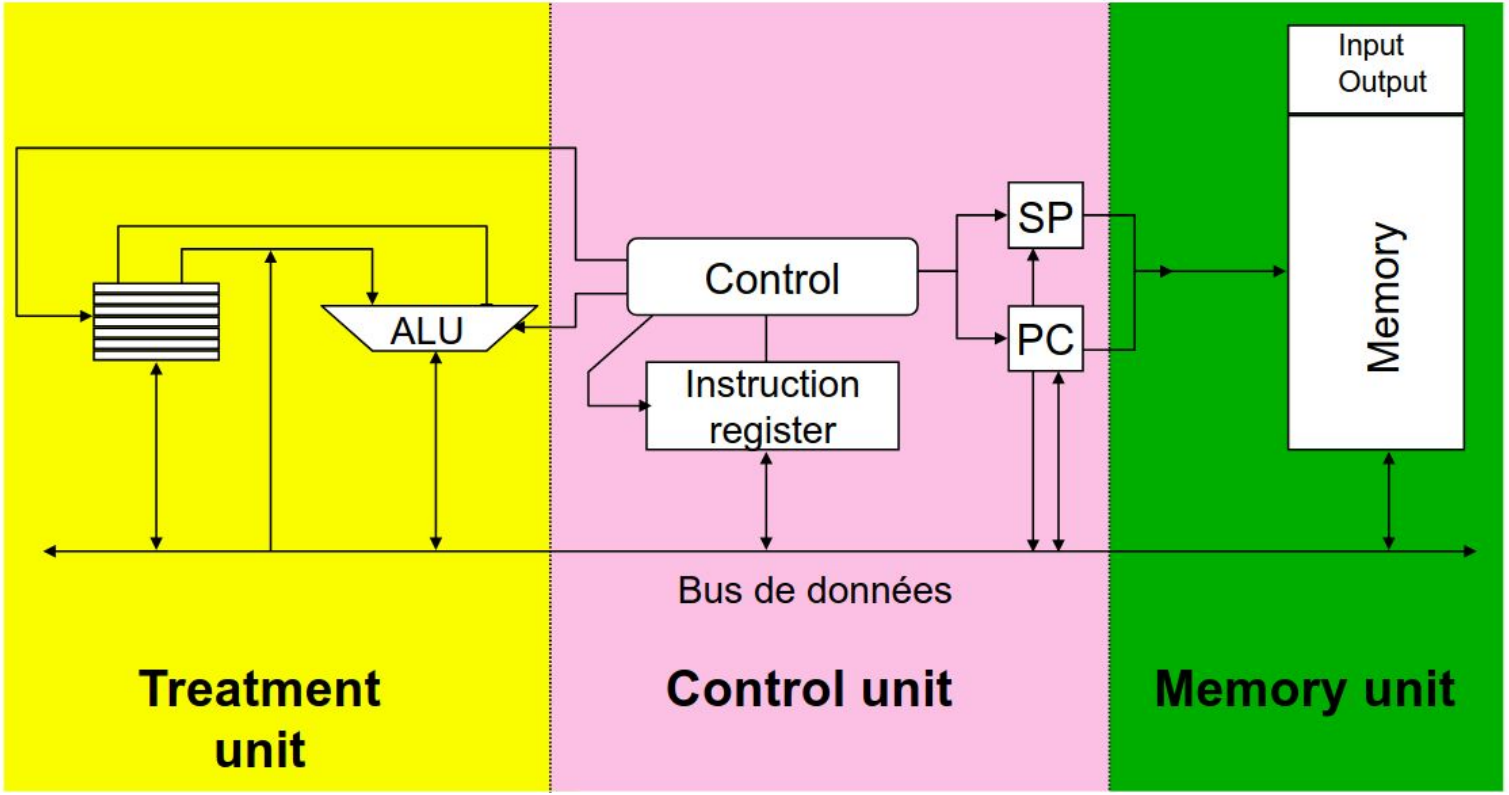
- Instruction execution
  - Decompose the instruction in a set of micro-instructions (set of signal to control) to ensure the correct execution of in the ALU
    - For example : imagine instruction ADD R1, R2, R3
    - This instruction needs control on R2 and R3, then control for ALU, then control for write register R1
  - Control the signals for the ALU
- Store the result
  - Store in memory or registers
  - Transfer data from register to memory
  - Store the stage of processor
    - Store the flags

Instruction executions: more detail

1. Load the next instruction from the memory to the instruction register
2. Modify the program counter (PC) for the next instruction to read
    a. PC = PC + 1
3. Decode the instruction
4. Find the operand necessary for the instruction
5. Load the operand in internal registers
6. Execute the instruction
7. Store the result in the memory
8. Go to the next instruction, return in step 1

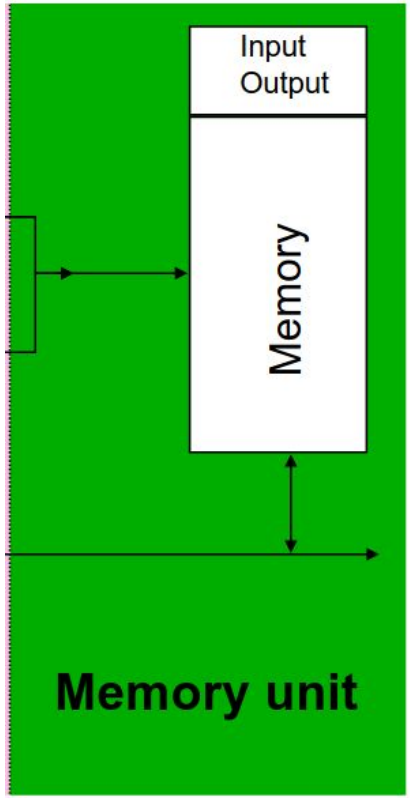
Von Neumann Designed Units

Von Neumann designed units



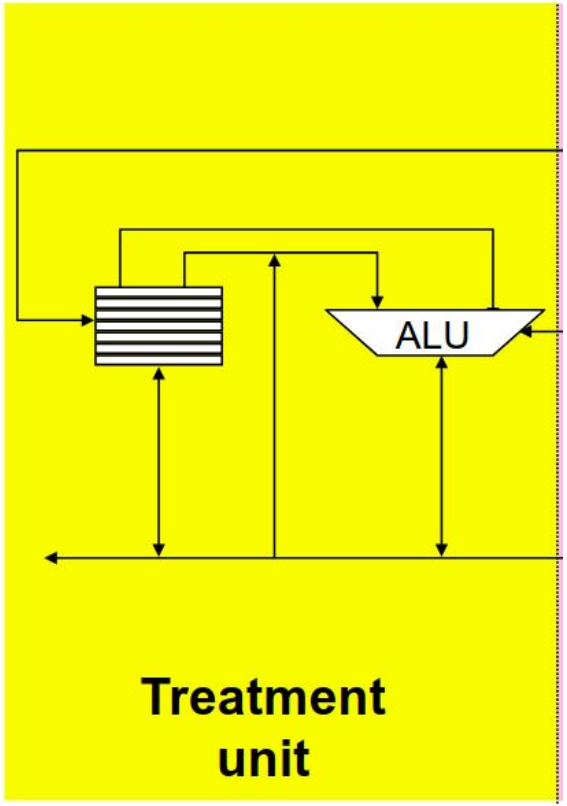
Memory unit

- Store the information which concern
  - The program, the application
  - The data to transform
- There are two types of memory model
  - Von Neumann model:
    - Instruction and data stored in single memory unit
    - Single access bus
  - Harvard model:
    - Instruction and data stored in different memory unit
    - Separate bus to access instruction and data



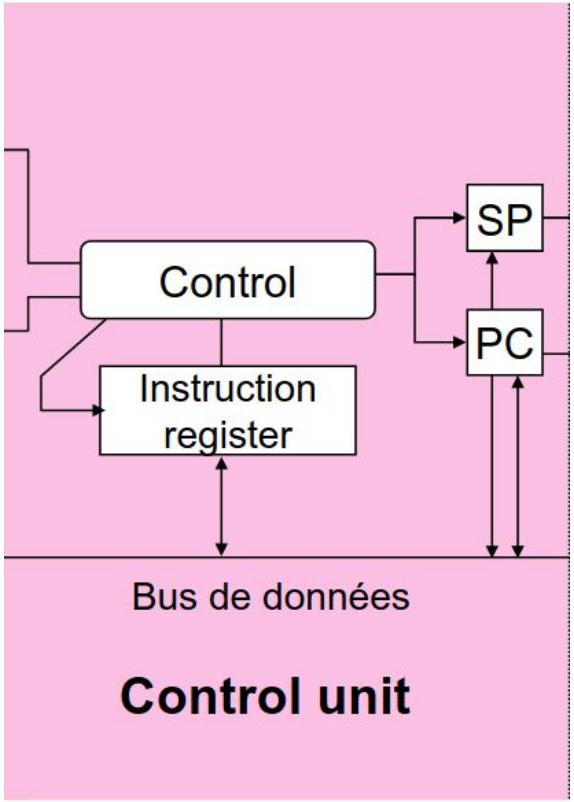
Treatment unit

- Composed of the ALU and a file of registers
- The registers store the data which are currently transformed by the ALU
  - Registers for data
  - Registers to store the state of the processor



Control unit

- The main part of the processor
- Ensure the execution of the instructions, of the program
  - Decode the instruction stored in the instruction register RI
  - Address the memory from the program counter
- Control execution signal
  - Interrupt signal
  - Instruct ALU, Memory, and IO device
- Hold instruction execution states
  - Stack Pointer (SP)
  - Program counter (PC)



Execution of the instructions

Three types of instructions

- Control instruction
  - Branch, jump, procedure call
- Treatment instruction
  - Arithmetic and logic
- Transfer instruction
  - Moving data
  - Between registers
  - Between memory and registers

Control / Branch Instruction

Control instructions

- Objective: modification of Program Counter (PC) depending on events
  - Internal events: results of some computation, state of processor
  - External events: I/O pads, interrupt
- The program counter is modified, and this change the sequence of addresses
- 3 types of jump/branch
  - **Conditional or unconditional**
  - **Explicit or implicit**
  - **With or without context saving**

Control instructions: conditional or unconditional branch

- Conditional
  - Result produces by a test, for example test of the flag values which indicates the state of the machine

- Unconditional
  - Jump/branch without any condition about the state of the processor

Control instructions: explicit or implicit branch

- Explicit
  - Case for a jump, or a conditional branch
- Implicit
  - Case for interrupt ⇒ External signal
  - Case for exceptions ⇒ For example, instruction which tries to divide by 0

Control instructions: jump/branch with context saving

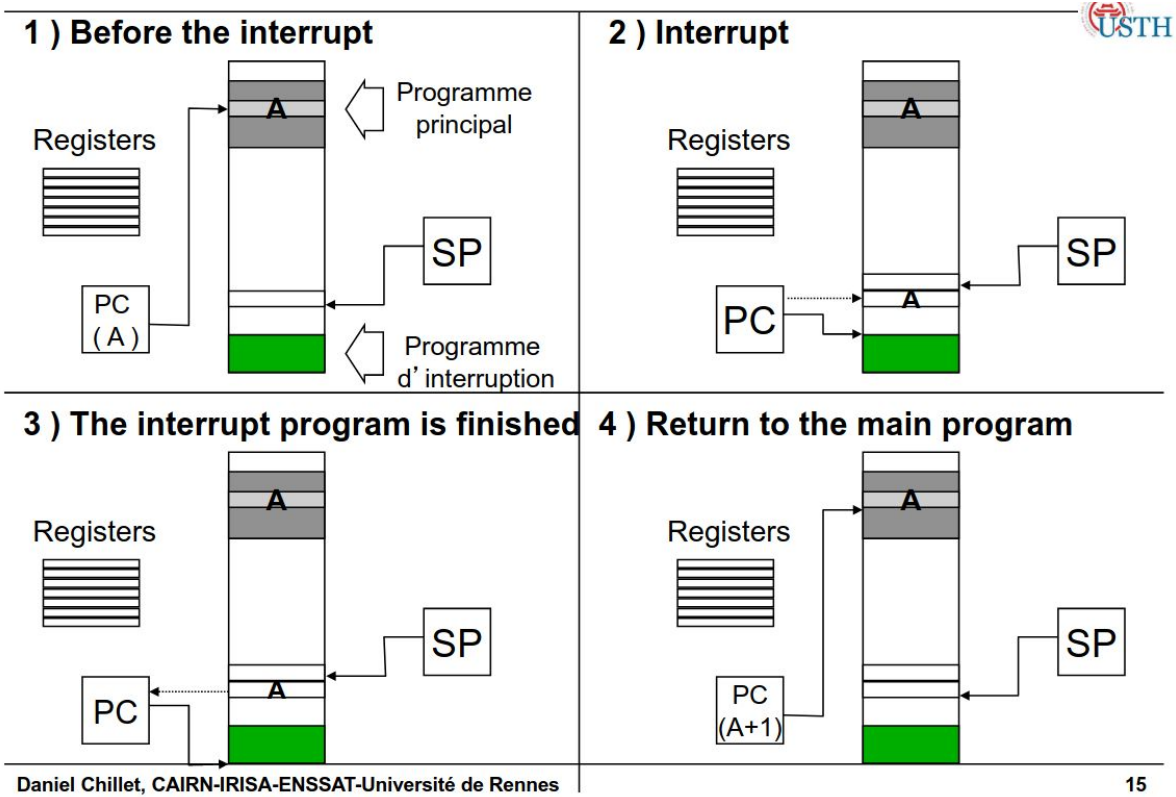
- The modification of PC due to an interruption or execution

  where the the current context of execution is stored

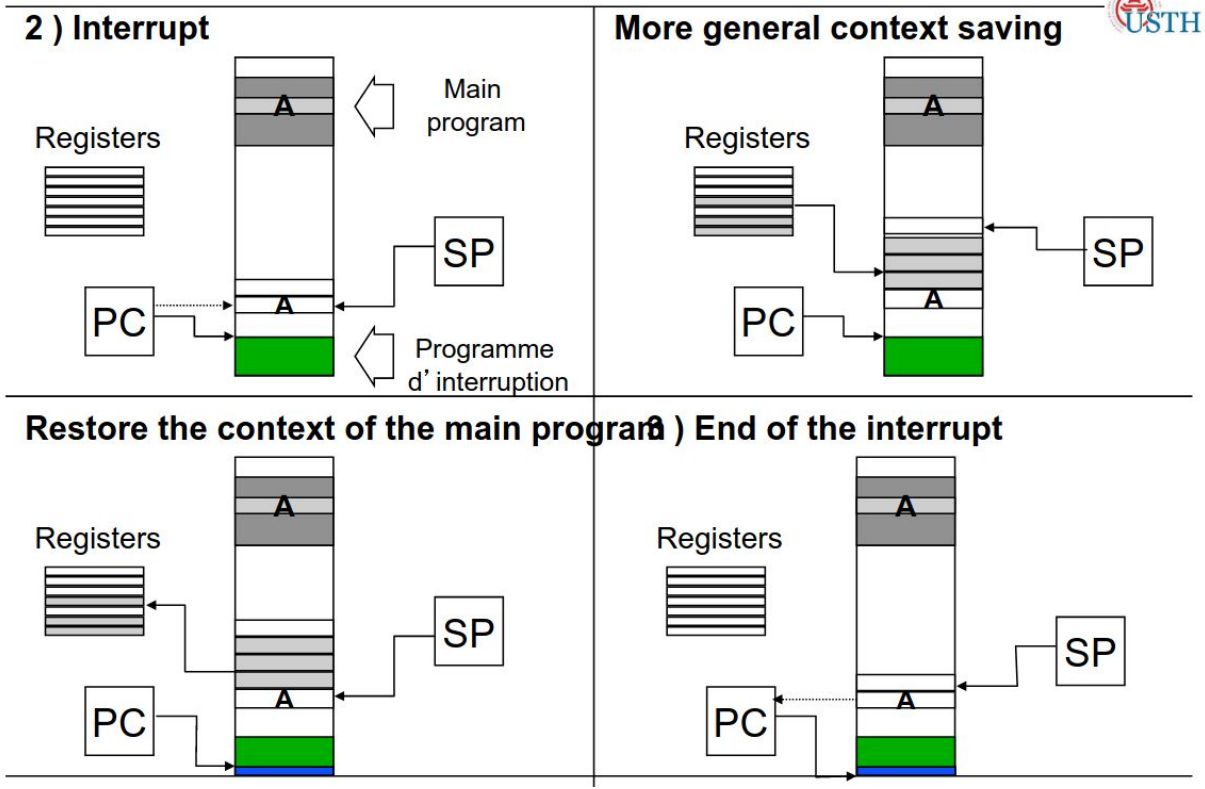
- The minimal context is composed of
  - The address of the current instruction
  - The content of the registers if needed ⇒ the software developer must

    explicitly specify them

Context saving: Memory State

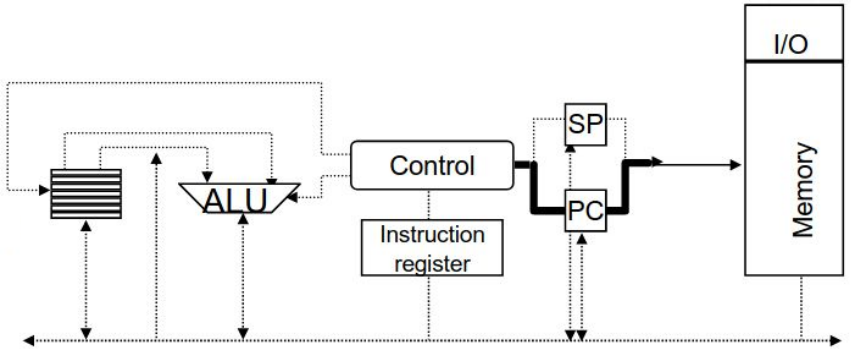


Context saving: Memory State (cont.)

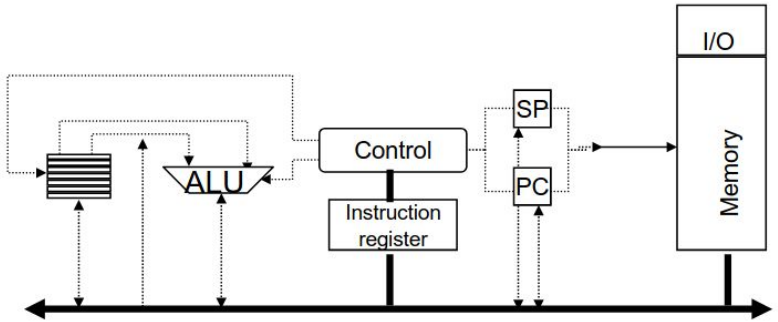


Control instruction: Organization

- Load instruction from memory

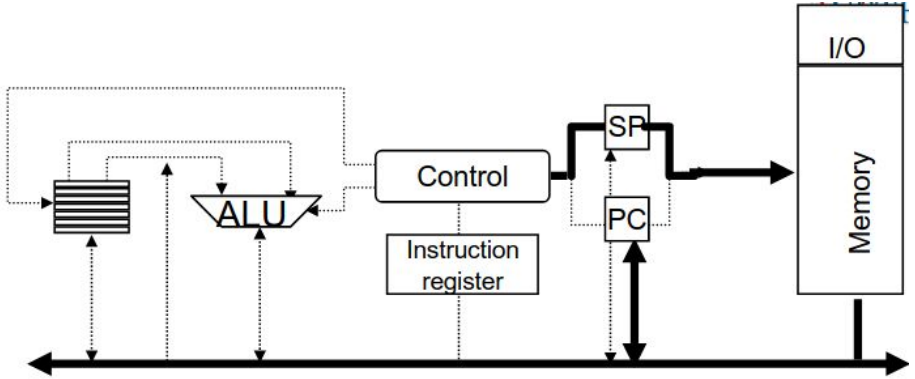


- Decode Instruction

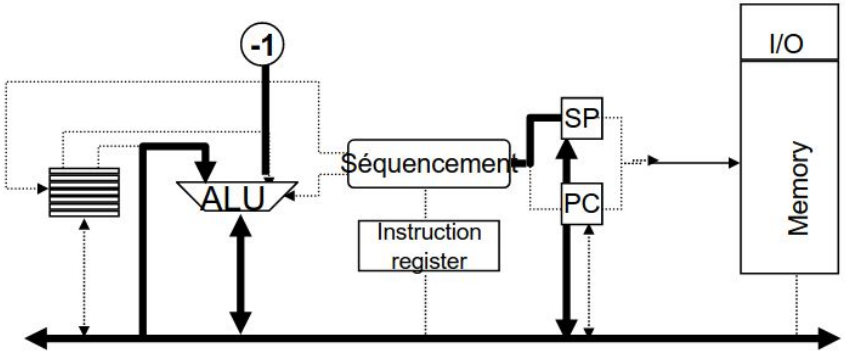


Control instruction: Organization (cont.)

- Context saving

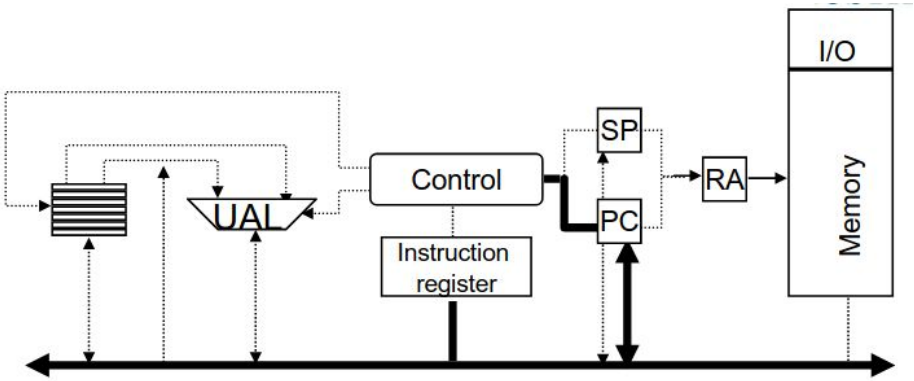


- Stack pointer modification



Control instruction: Organization (cont.)

- Extract the branch address



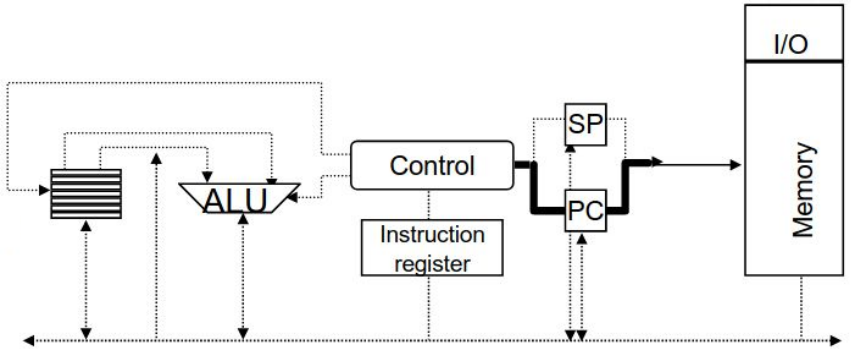
Treatment / Computation Instruction

Data treatment

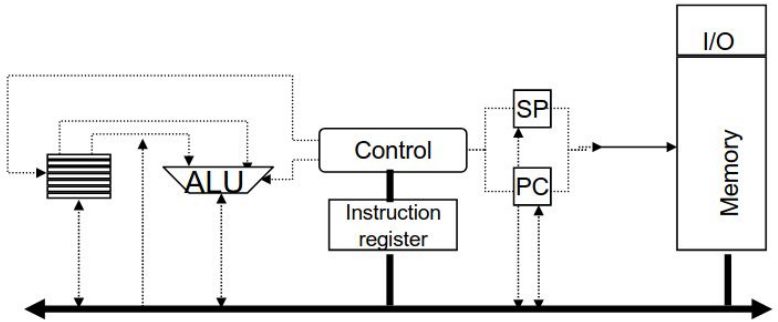
- Computation operations
  - Arithmetic
  - Logic
- Modification of the flags depending on the results produced by ALU
  - Result is equal to 0
  - Result is negative
  - Result is impossible to code with the classical number of bits, carry

Treatment instruction: Organization

- Load instruction from memory

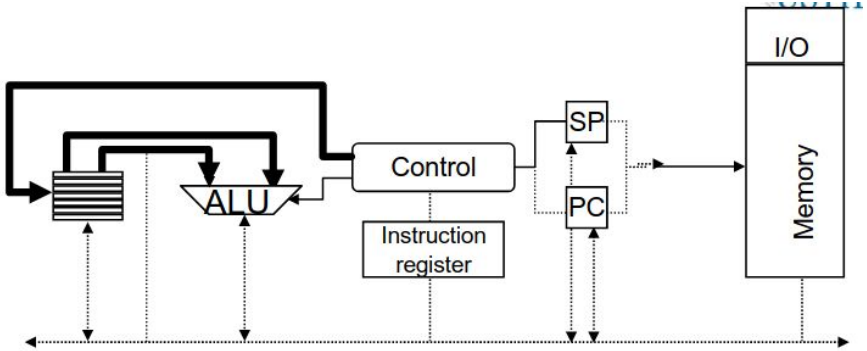


- Decode Instruction

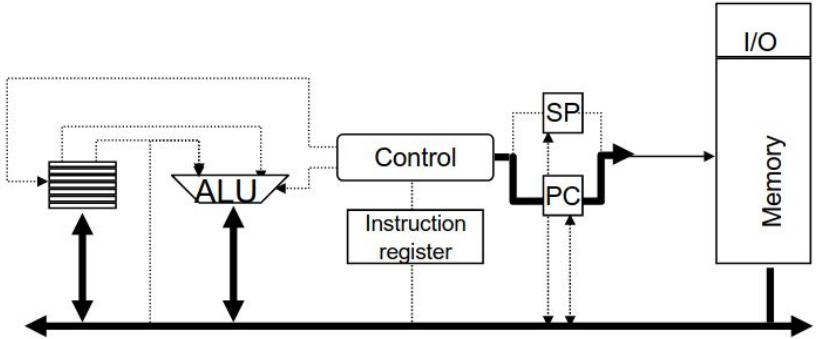


Treatment instruction: Organization (cont.)

- Instruction execution

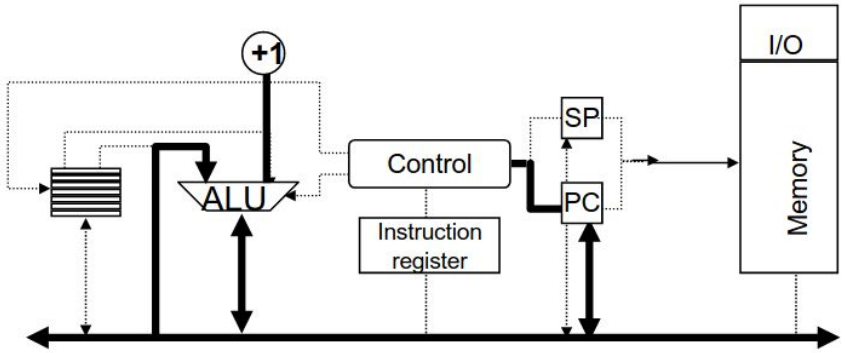


- Result storage



Treatment instruction: Organization (cont.)

- Go the next address, Increment the counter program



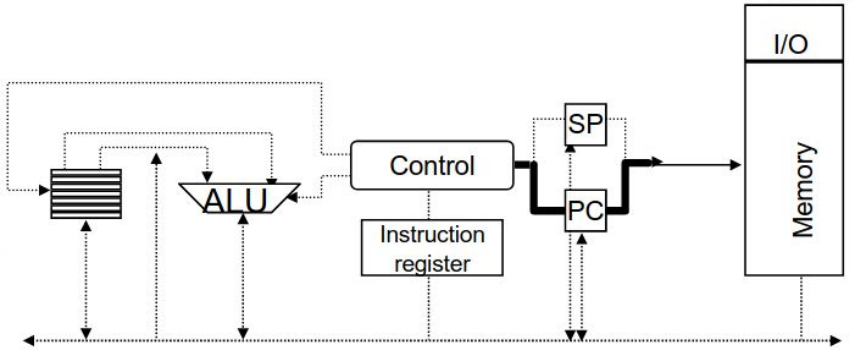
Transfer Instruction

Transfer Instructions

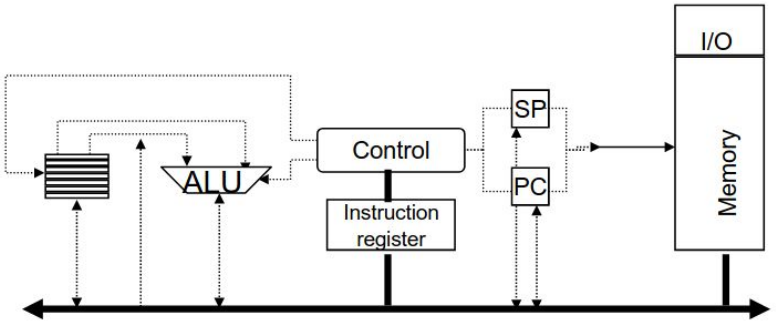
- Move data between different placements
- There are three types of transfers
  - **Between registers**
  - **Between register and memory**
  - Between two differents cells of memory
- To address a data, different techniques
  - Immediate addressing
  - Direct addressing
  - Indirect addressing
  - Indexed addressing
  - etc.

Transfer instruction: Organization

- Load instruction from memory

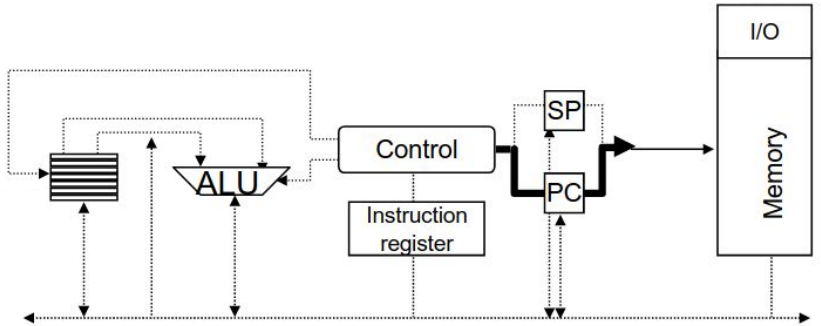


- Decode Instruction

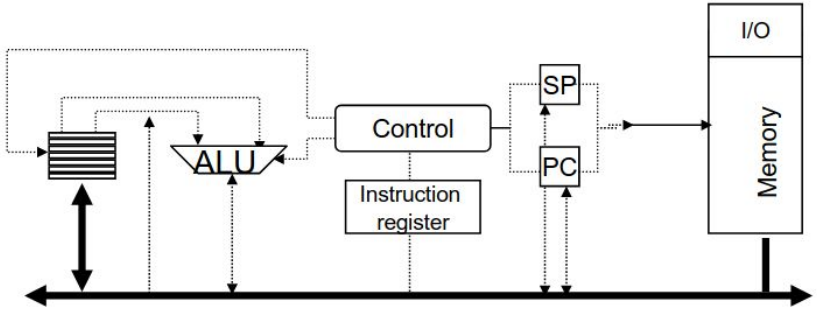


Transfer instruction: Organization (cont.)

- Looking for the data to transfert



- Transfer effective



Memory Addressing

Memory addressing

- Data is located at different locations
  - Registers
  - Memory
  - Instruction itself
  - ⇒ Need method to specify which data to process
- Effective Address (EA)
  - Final address of the operand (data) in the instruction
  - Reveals the location of operand
- Address Mode
  - Specifies how to calculate the Effective Address of the operand(s) from the instruction
  - Syntax: depending on the programming language

Memory addressing (cont.)

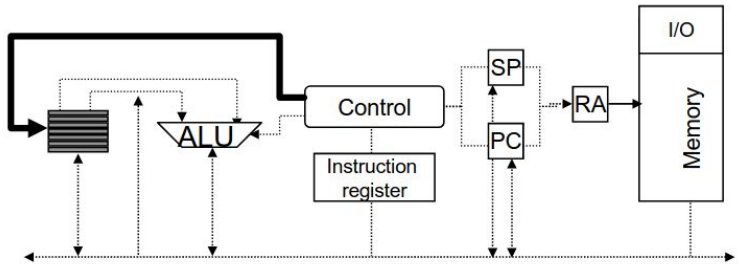
- **Register addressing - Transfer data inside register file**
- **Immediate addressing - Data is located inside instruction itself**
- **Direct addressing - Operand is a pointer point to data location**
- **Indirect addressing - Operand is a pointer of pointer to data location**
- **Indexed addressing - Calculate data location from (RI + based index)**
- Post / Pre increment indirect addressing - Same like (i++ / i–) in C

Register addressing

- Instruction coding

- Instruction examples:
  - add R1, R2, R3 ; R1 = R2 + R3
  - move R1, R2 ; R1 = R2

- Step of the instruction
  - Load instruction from memory
  - Decode instruction
  - Transfer effective

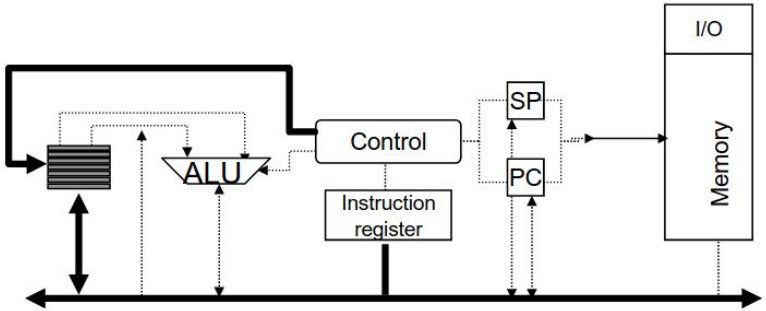


Immediate addressing

- Instruction coding

- Instruction examples:
  - add R1, R2, #3 ; R1 = R2 + 3
  - load R1, #20 ; R1 = 20

- Step of the instruction
  - Load instruction from memory
  - Decode instruction
  - Transfer effective



Direct addressing

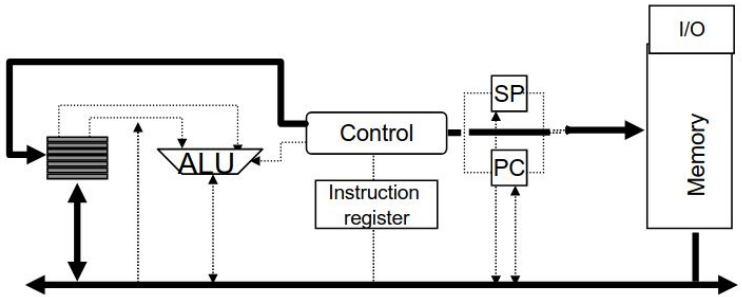
- Instruction coding

code	Destination register	Word 1
Part 1 of the address		Word 2
Part 2 of the address		Word 3

- Instruction examples:
  - add R1, R2, (1000) ; R1 = R2 + Memory[1000]
  - li R4, (2000) ; R4 = Memory[2000]

- Type of data: variable, constant

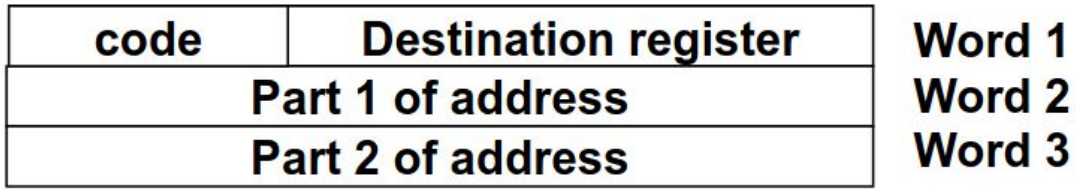
- Step of the instruction
  - Load instruction from memory
  - Decode instruction
  - Looking for the data to transfert
  - Transfer effective



Indirect addressing

- Instruction coding: indirect through register

- Instruction coding: indirect through memory



- Instruction examples:
  - Add R1, R2, @(R3) ; R1 = R2 + Memory[Memory[R3]]
  - Li R4, @(R5) ; R4 = Memory[Memory[R5]]

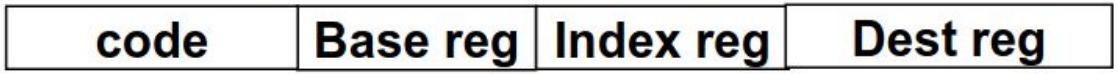
Indirect addressing

- Type of data: variable, array, pointer, constant
- Step of execution
  - **Indirect register:**
    - Load instruction from memory (A)
    - Decode instruction (B)
    - Looking for the data to transfert (C)
    - Transfer effective (D)
  - **Indirect memory:**
    - (A, B) * 3, (C, D) * 3

Indexed addressing

- Instruction coding: immediate indexing

- Instruction coding: Indexed through register



- Instruction example:
  - Add R1, R2, R3(R4) ; R1 = R2 + Memory[R3+R4]
- Types: variable, data structure

Indexed addressing (cont.)

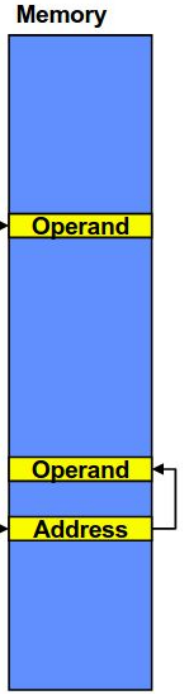
- Step of execution
  - **Indexation through register:**
    - Load instruction from memory (A)
    - Decode instruction (B)
    - Looking for the data to transfert (C)
    - Transfer effective (D)
    - Addition of base and index
  - **Immediate indexation:**
    - (A, B) * 2, C, D, addition of index and base

To resume

- Immediat

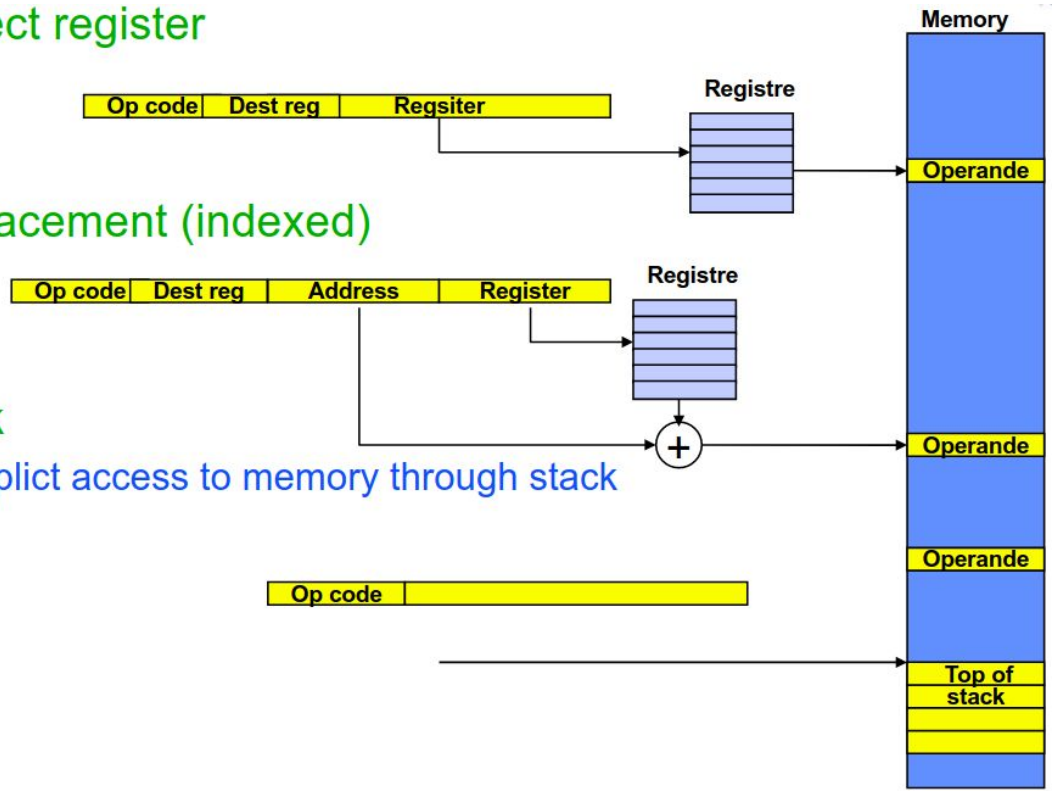
- Direct

- Indirect through memory



To resume (cont.)

- Indirect register
- Displacement (indexed)
- Stack
  - Implict access to memory through stack



ISA Extra Features

Size of instruction set

- **CISC processor:**
  - The number of words to code the instructions is variable

- **RISC processor:**
  - Each instruction is coded with fix size of word

- **Discussion: RISC make more simpler controller**

Data addressing methods

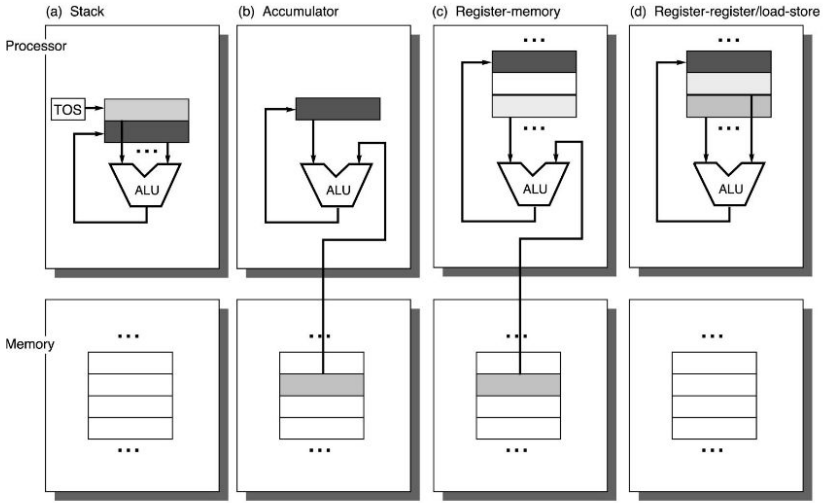
- **Memory based:** no register available for user, all the operations must be done with a memory cell
- **Accumulator based:** all the operations are executed with a specific register (the accumulator)
- **Register based:** all the operations are executed with the register file
- **Stack based:** the operations are stored in a stack

Data addressing methods (cont.)

Stack	Accumulator	Register-Memory	Load-store

Push A
Push B
Add
Push A
Mul
Pop C
	Load A
	Add B
	Mul A
	Store C
		Load R1,A
		Add R1,B
		Mul R1,A
		Store C,R1
			Load R1,A
			Load R2,B
			Add R3,R1,R2
			Mul R3,R3,R1
			Store C,R3

6 instr. 4 mem. op.	4 instr. 4 mem. op.	4 instr. 4 mem. op.	5 instr. 3 mem. op.



Orthogonal instructions and number of operands

- A computer instruction set architecture that:
  - No dependencies between operation code and addressing modes
  - All operations can used all the addressing modes
  - In general, these processors have small number of instructions
  - These processor are simple to program
  - Compiler easier to develop
- General number of operands in an instruction
  - 0 operand (NOP)
  - 1 operands
  - 2 operands (instructions modify one operand)
  - 3 operands

Thank you for you listening