Unit 2. CPU

Computer Architecture

Area of Computer Architecture and Technology
Department of Computer Science and Engineering
University of Oviedo

Fall, 2015

Objectives

1.- Support for multitasking operating systems

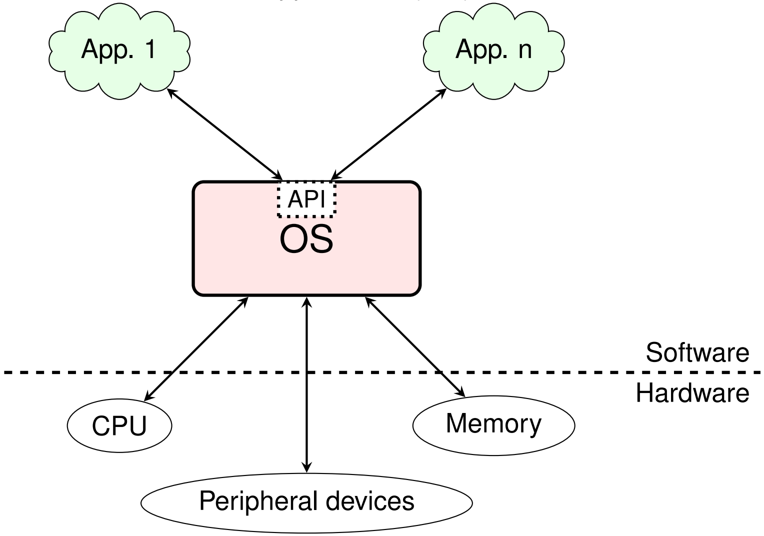
Requirements to deal with this task

2.- Performance improvements

Organizational changes to improve performance

Multitasking operating system

Common interface for applications (API)



Multitasking

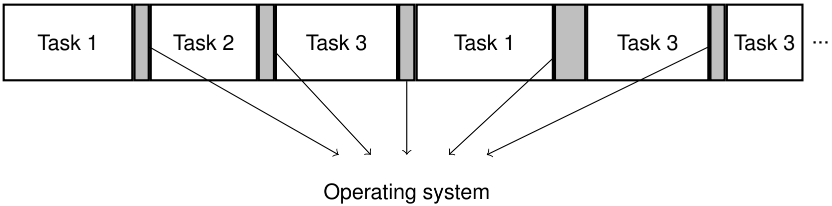
What is it?

Ability to run several tasks 'simultaneously'

- Downside: Resources are limited

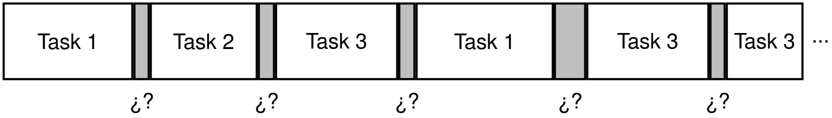
Solution

The OS manages the resources and shares them among tasks



If the time slice (quantum) assigned for each task is small, apparent concurrency is achieved

Scheduling techniques



The OS acts as a resource manager

In which conditions is the control transferred to the OS?

- Service calls from tasks
- Interrupts (from tasks or from I/O)
- Exceptions

Scheduling techniques

- Each condition implies executing a different handler
- The handler may modify the state of the tasks
- The scheduling techniques can be nested
  - Example: an exception inside an interrupt
- Architecture dependent naming

If none condition is reached?

Add timer $\Rightarrow$ The OS takes control of the processor periodically

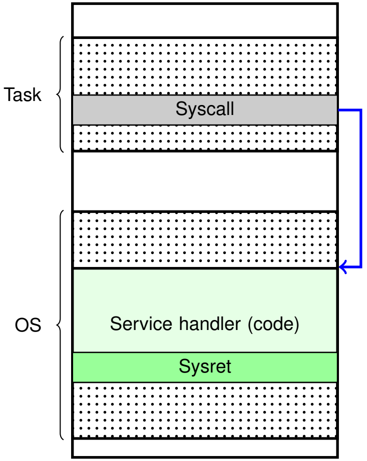
Service call to the OS

What is it?

A task issues a service call to the OS. Example: open a file

Procedure

1. The task invokes *syscall*
2. The OS executes the service handler
3. *Sysret* makes the processor return to the instruction that follows *syscall*



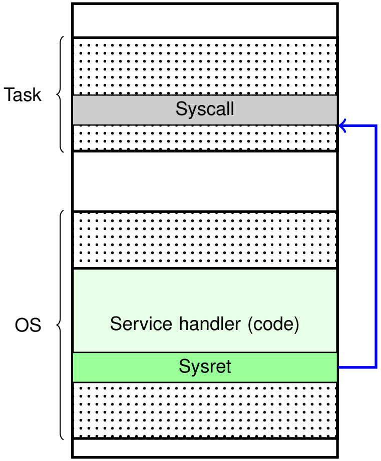
Service call to the OS

What is it?

A task issues a service call to the OS. Example: open a file

Procedure

1. The task invokes *syscall*
2. The OS executes the service handler
3. *Sysret* makes the processor return to the instruction that follows *syscall*



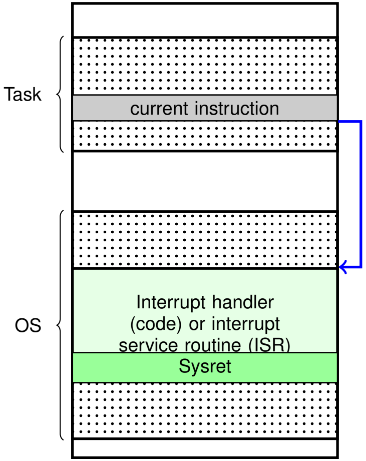
Interrupt

What is it?

The OS receives and processes a request from I/O

Procedure

1. The interrupt request line is activated
2. The current instruction is executed until its end
3. The OS executes the interrupt handler
4. *Sysret* makes the processor return to the following instruction of the task



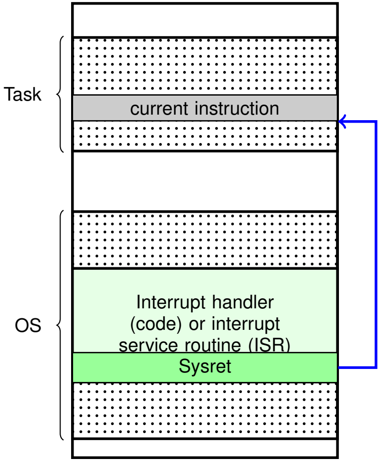
Interrupt

What is it?

The OS receives and processes a request from I/O

Procedure

1. The interrupt request line is activated
2. The current instruction is executed until its end
3. The OS executes the interrupt handler
4. *Sysret* makes the processor return to the following instruction of the task



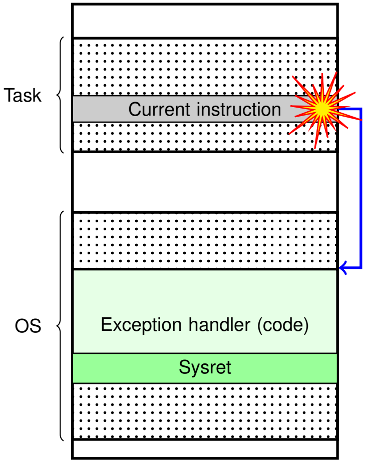
Exception

What is it?

The OS takes control of the processor in the presence of an anomalous situation while executing an instruction

Procedure

1. An anomalous situation occurs. Example: Divide by zero
2. The current instruction is NOT finished
3. The OS executes the exception handler
4. Different return scenarios may occur



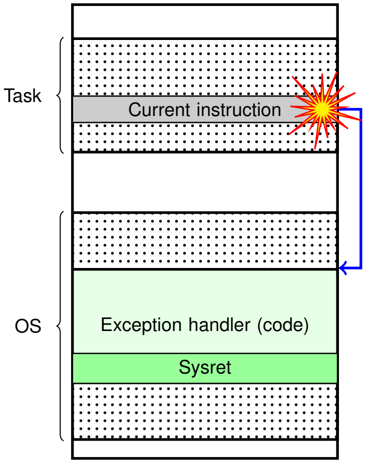
Exception

What is it?

The OS takes control of the processor in the presence of an anomalous situation while executing an instruction

Procedure

1. An anomalous situation occurs. Example: Divide by zero
2. The current instruction is NOT finished
3. The OS executes the exception handler
4. Different return scenarios may occur



Multitasking OS in the CT

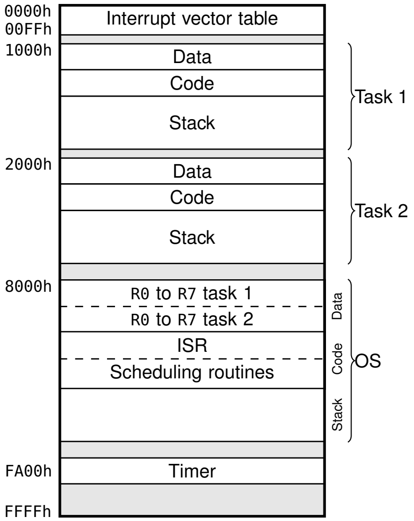
Objective

Determine CPU limitations

- A timer requests interrupts periodically

CPU registers

Task 1 values



Multitasking OS in the CT

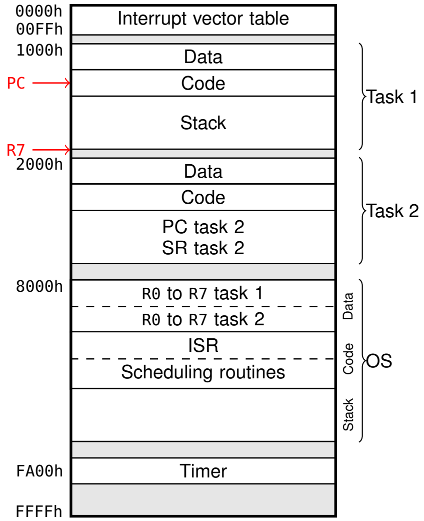
Step 1

Task 1 is running

- PC & R7 reference task 1
- CPU registers contain values/addresses of task 1

CPU registers

Task 1 values



Multitasking OS in the CT

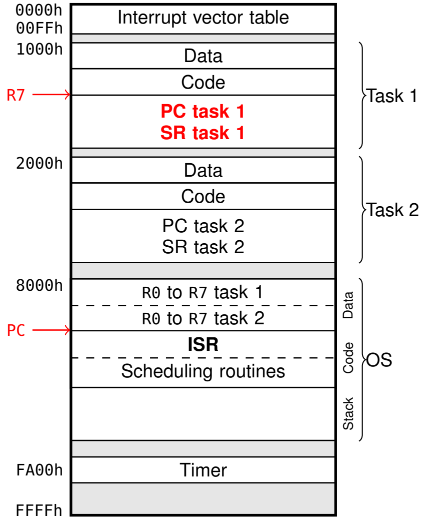
Step 2

Timer interrupt is requested

- SR & PC are pushed in the stack
- PC changes ⇒ the OS takes the control

CPU registers

Task 1 values



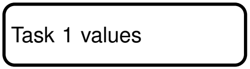
Multitasking OS in the CT

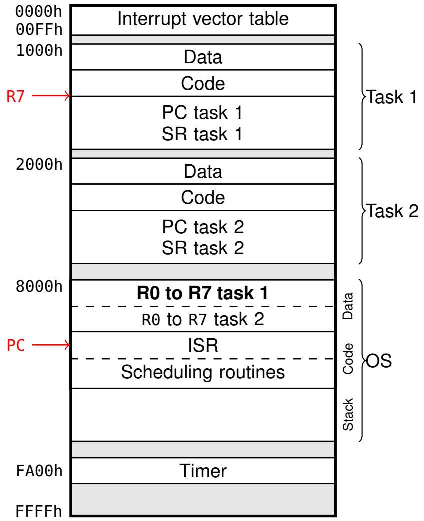
Step 3

The OS stores the state of task 1

- The OS copies CPU register values in its data memory

CPU registers





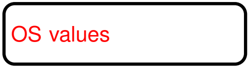
Multitasking OS in the CT

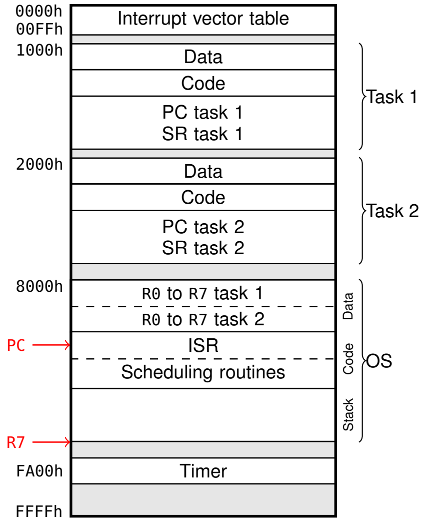
Step 4

The OS switches the stack

- CPU registers are available to the OS
- The stack pointer register references the OS stack

CPU registers





Multitasking OS in the CT

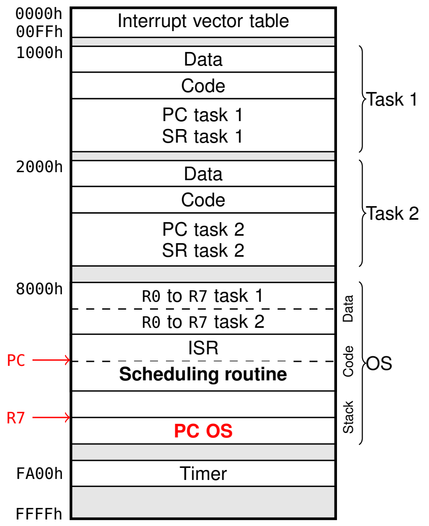
Step 5

The OS is running

- Update statistics, system clock, etc.
- Runs the scheduler

CPU registers

OS values



Multitasking OS in the CT

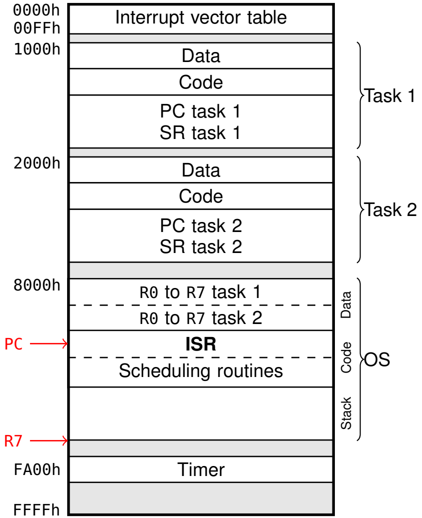
Step 6

The OS is still running

- The scheduler returns
- Next task to run is task 2

CPU registers

OS values



Multitasking OS in the CT

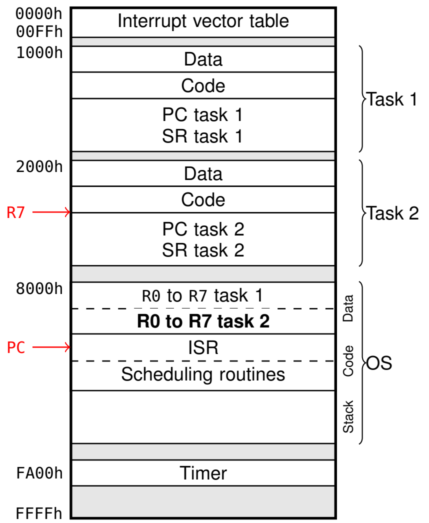
Step 7

The OS starts scheduling task 2

- Task 2 registers are restored (R7 included)
- The stack is switched

CPU registers

Task 2 values



Multitasking OS in the CT

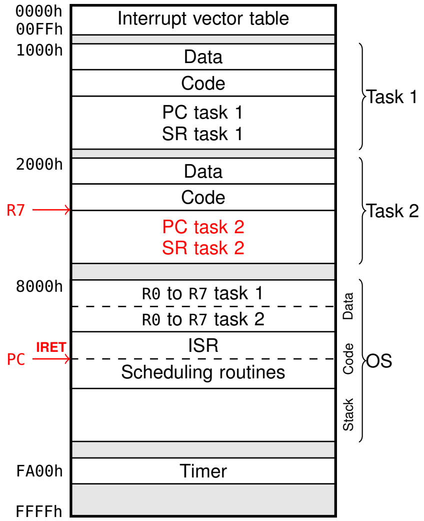
Step 8

The OS finishes transferring the control to task 2

- `IRET` is executed (last instruction of the routine)

CPU registers

Task 2 values



Multitasking OS in the CT

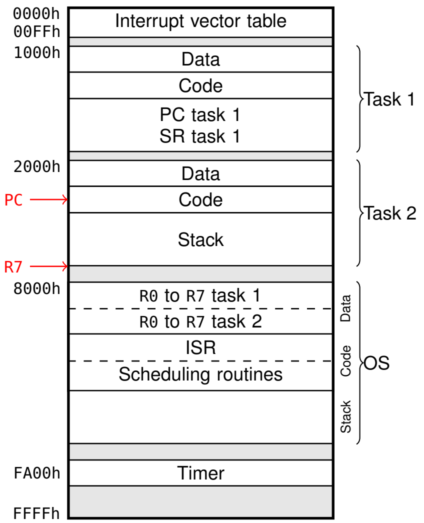
Step 9

Task 2 is running

- Its execution starts from where it was previously paused

CPU registers

Task 2 values



Multitasking OS in the CT

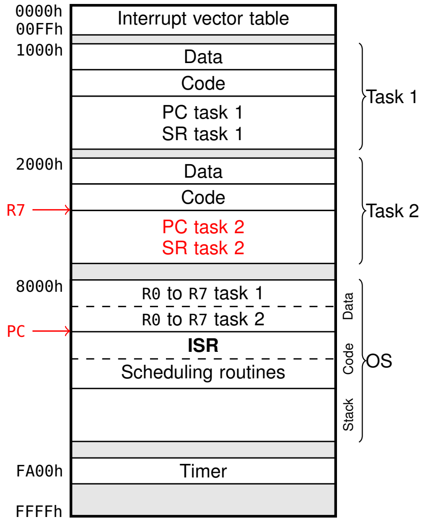
Step 10

New timer interrupt

- The procedure is repeated again

CPU registers

Task 2 values



CT deficiencies

Uncontrolled execution of instructions

- A task can execute any instruction
- Even disabling interrupts (`CLI`)
- There is no task switching

Solution

Establish privilege levels over the instruction set

- Add one bit to the status register showing the current privilege level

CT deficiencies

Uncontrolled execution of instructions

- A task can execute any instruction
- Even disabling interrupts (`CLI`)
- There is no task switching

Solution

Establish privilege levels over the instruction set

- Add one bit to the status register showing the current privilege level

CT deficiencies

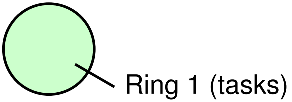
Uncontrolled execution of instructions

- A task can execute any instruction
- Even disabling interrupts (`CLI`)
- There is no task switching

Solution

Establish privilege levels over the instruction set

- Add one bit to the status register showing the current privilege level



CT deficiencies

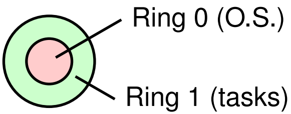
Uncontrolled execution of instructions

- A task can execute any instruction
- Even disabling interrupts (`CLI`)
- There is no task switching

Solution

Establish privilege levels over the instruction set

- Add one bit to the status register showing the current privilege level



CT deficiencies

Uncontolled access to the OS memory address range

- A task can access the memory locations of the OS
- It can also access I/O interfaces and system tables

Solution

Task access to OS memory range must be restricted

- Address ranges accesses require a specific privilege level

CT deficiencies

Uncontolled access to the OS memory address range

- A task can access the memory locations of the OS
- It can also access I/O interfaces and system tables

Solution

Task access to OS memory range must be restricted

- Address ranges accesses require a specific privilege level

CT deficiencies

Uncontrolled access to memory address ranges of other tasks

- A task can access memory locations of other tasks

Solution

The address ranges accessible by a task must be restricted

- Data structure specifying the range allowed

Memory protection

- Protect the OS against the tasks
- Protect the tasks among themselves

CT deficiencies

Uncontrolled access to memory address ranges of other tasks

- A task can access memory locations of other tasks

Solution

The address ranges accessible by a task must be restricted

- Data structure specifying the range allowed

Memory protection

- Protect the OS against the tasks
- Protect the tasks among themselves

CT deficiencies

Using the stack of the task in the scheduling process

- A task can assign any value to the stack pointer
- The stack of the task can overflow or underflow

Solution

Use the OS stack only. Two options:

1. Hardware stack switching
2. Auxiliary registers to store SR & PC

CT deficiencies

Using the stack of the task in the scheduling process

- A task can assign any value to the stack pointer
- The stack of the task can overflow or underflow

Solution

Use the OS stack only. Two options:

1. Hardware stack switching
2. Auxiliary registers to store SR & PC

CT deficiencies

There is no exception management

- Anomalous situations while executing instructions is not considered
- Wrong machine code

Solution

Manage exceptions as interrupts

- Associate an interrupt vector number for each exception

CT deficiencies

There is no exception management

- Anomalous situations while executing instructions is not considered
- Wrong machine code

Solution

Manage exceptions as interrupts

- Associate an interrupt vector number for each exception

Parameters of a CPU

Basic

- CPU width
    - # of bits of the operands
- Addressable memory (m)
    - # of lines in the SAB (a) $\Rightarrow m = 2^a$
- Memory word size
    - # of lines in the SDB
- Register set
- Instruction set
- Addressing modes
- *Endianity*

Functional units

- Datapath
    - registers, ALU, FPU, buses
- Control unit

Parameters of a CPU

Basic

- CPU width
  - # of bits of the operands
- Addressable memory (m)
  - # of lines in the SAB (a) $\Rightarrow m = 2^a$
- Memory word size
  - # of lines in the SDB
- Register set
- Instruction set
- Addressing modes
- *Endianity*

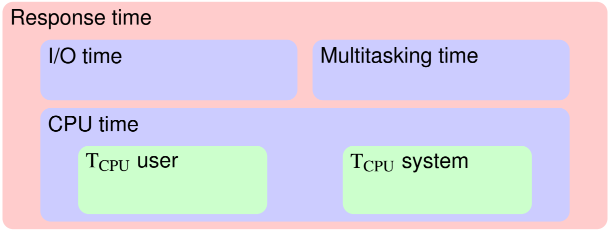
Functional units

- Datapath
  - registers, ALU, FPU, buses
- Control unit

Performance improvements

CPU time reduction

Reduce the response time of the programs



The speedup is computed taking the Amdahl's law into account

- It depends on the percentage of use of the CPU

Performance improvements

$$\boxed{T_{\text{CPU}} = \#\text{ of instructions} \times \text{CPI} \times T}$$

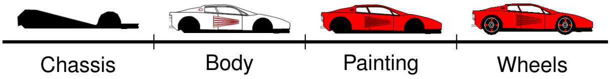
Reducing the CPU time requires to cut down

1. # of (assembly) instructions of the program
   - reducing this number requires improving the instruction set and/or the compiler
2. Clock rate
   - increasing the clock frequency requires improving the manufacturing technology
3. CPI
   - reducing the average number of cycles per instruction requires segmenting the execution of instructions and/or replacing functional units of the CPU

Segmentation of instructions (pipelining)

Simile

It resembles an assembling line



When a stage is completed, the product moves to the next one

Key

Divide the execution of instructions in stages that can run in parallel

Segmentation of instructions (pipelining)

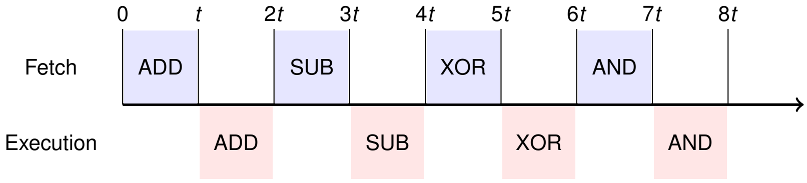
Example in the CT

The instructions are divided in two stages of the same duration:

- instruction fetch
- execution

```
ADD R4, R3, R2
SUB R2, R3, R4
XOR R5, R5, R5
AND R3, R4, R5
```

The stages run in a sequential-execution mode



Segmentation of instructions (pipelining)

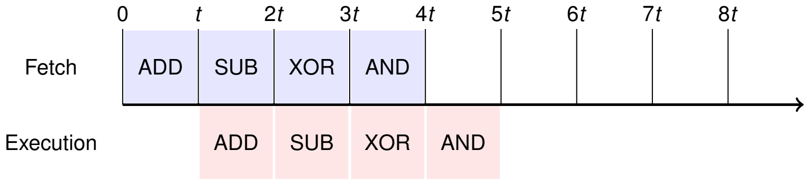
Example in the CT

The instructions are divided in two stages of the same duration:

- instruction fetch
- execution

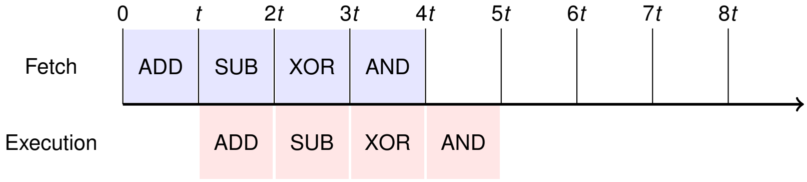
```
ADD R4, R3, R2
SUB R2, R3, R4
XOR R5, R5, R5
AND R3, R4, R5
```

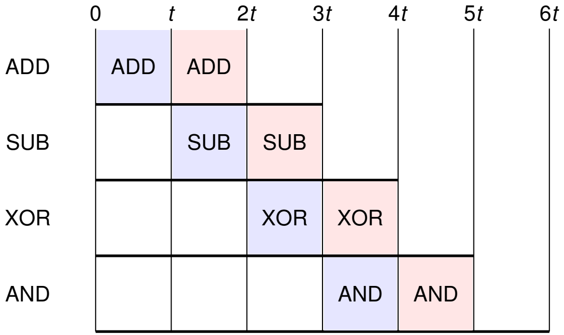
Now the stages run in parallel



Segmentation of instructions (pipelining)

It can also be represented as follows





Segmentation of instructions (pipelining)

How much time does it take to execute an instruction?

Response time measurement (latency of the pipeline)

How often does the CPU complete an instruction?

Throughput measurement

Ideal scenario

The pipeline increases the throughput by a factor equal to the number of stages, without modifying the clock frequency of the CPU

- Downside: the complexity of the control unit of the CPU increases

Segmentation hazards

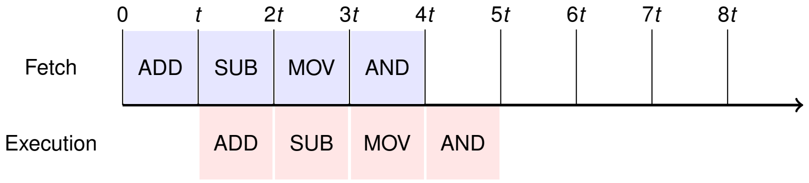
Example in the CT

The instructions are divided in two stages of the same duration:

- instruction fetch
- execution

```
ADD R4, R3, R2
SUB R2, R3, R4
MOV R3, [R2]
AND R3, R4, R5
```

In an ideal scenario



Segmentation hazards

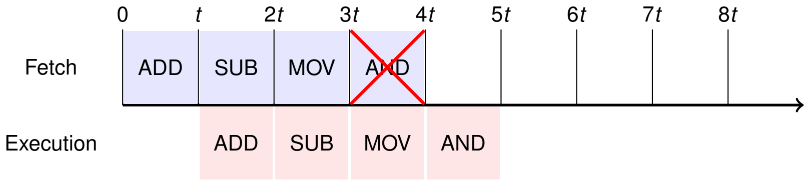
Example in the CT

The instructions are divided in two stages of the same duration:

- instruction fetch
- execution

```
ADD R4, R3, R2
SUB R2, R3, R4
MOV R3, [R2]
AND R3, R4, R5
```

In a real scenario the processor may deal with hazards by stalling and creating a bubble in the pipeline



Segmentation hazards

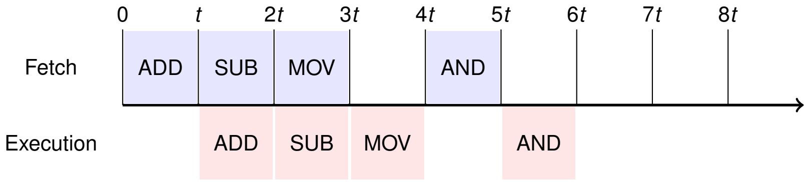
Example in the CT

The instructions are divided in two stages of the same duration:

- instruction fetch
- execution

```
ADD R4, R3, R2
SUB R2, R3, R4
MOV R3, [R2]
AND R3, R4, R5
```

In a real scenario the processor may deal with hazards by stalling and creating a bubble in the pipeline



Segmentation of instructions (pipelining)

Conclusions

- Segmentation increases the throughput of the CPU while changing its internal organization
- The upper limit of the throughput is given by the number of stages
- Segmentation hazards are present
    - accesses to shared resources, branches, interrupts, etc.
- Implies a complexity increase in the control unit of the CPU

Functional units replication

Avoid structural hazards (unique resources)

Replicate functional units that can provoke stalls in the pipeline

- ALU, internal bus, memory interface, etc.
- Downside: the complexity of the control unit increases

Superscalar CPU

Pipelined CPU with replication of functional units that can execute more than one instruction per cycle (several pipeline channels)

- Extreme scenario: replicate the whole processor $\Rightarrow$ multiprocessor

Thread-level parallelism

A thread is the basic unit for scheduling in an OS

Process

It consists of at least one thread

- it may have several threads

$\boxed{\text{Thread-level parallelism}} \neq \boxed{\begin{array}{c}\text{Instruction-level parallelism}\\ \text{(\textit{pipeline} or superscalar architectures)}\end{array}}$

Implications

- Instruction-level parallelism improves the execution of any program
  - it is transparent for the programmer
- Thread-level parallelism improves the execution of some programs
  - the programmer must take active part: thread creation & synchronization, etc.

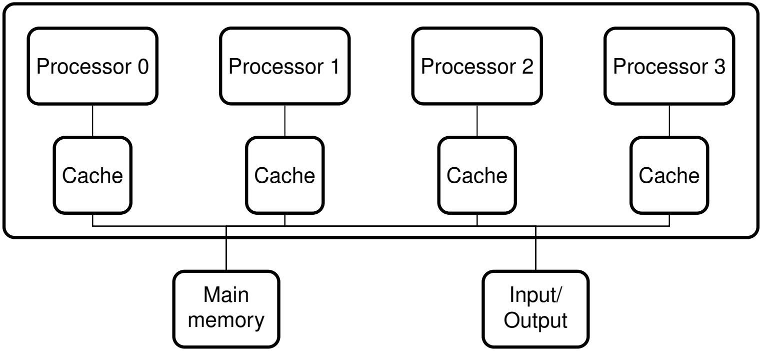
Flynn’s taxonomy

Classification of computer architectures based on the number of concurrent instructions and data flows

1. SISD: single instruction, single data
   - sequential computer
2. SIMD: single instruction, multiple data
   - GPU, MMX/SSE (multimedia) instructions
3. MISD: multiple instructions, single data
   - fault tolerant applications
4. MIMD: multiple instructions, multiple data
   - Distributed memory: isolated processing units connected by a network
   - Shared memory: groups of processing and memory units

Multicore systems

Nowadays computers are MIMD systems with shared memory (*Shared Memory Multiprocessors*)



Simultaneos Multithreading (HyperThreading)

Replication of functional units to execute two threads per core

- Downside: cache-memory access

Unit 2. CPU

Computer Architecture

Area of Computer Architecture and Technology
Department of Computer Science and Engineering
University of Oviedo

Fall, 2015