MMP16: Didactic Micro-Programmed Micro-Processor

José Luis López Presa Elio Pérez Calle

Department of Telematic Engineering and Architecture,
Universidad Politécnica de Madrid, Spain (UPM)
jllopez@diatel.upm.es

EU Science and Technology Fellowship Programme,
University of Science and Technology of China (USTC)
elio@ustc.edu.cn

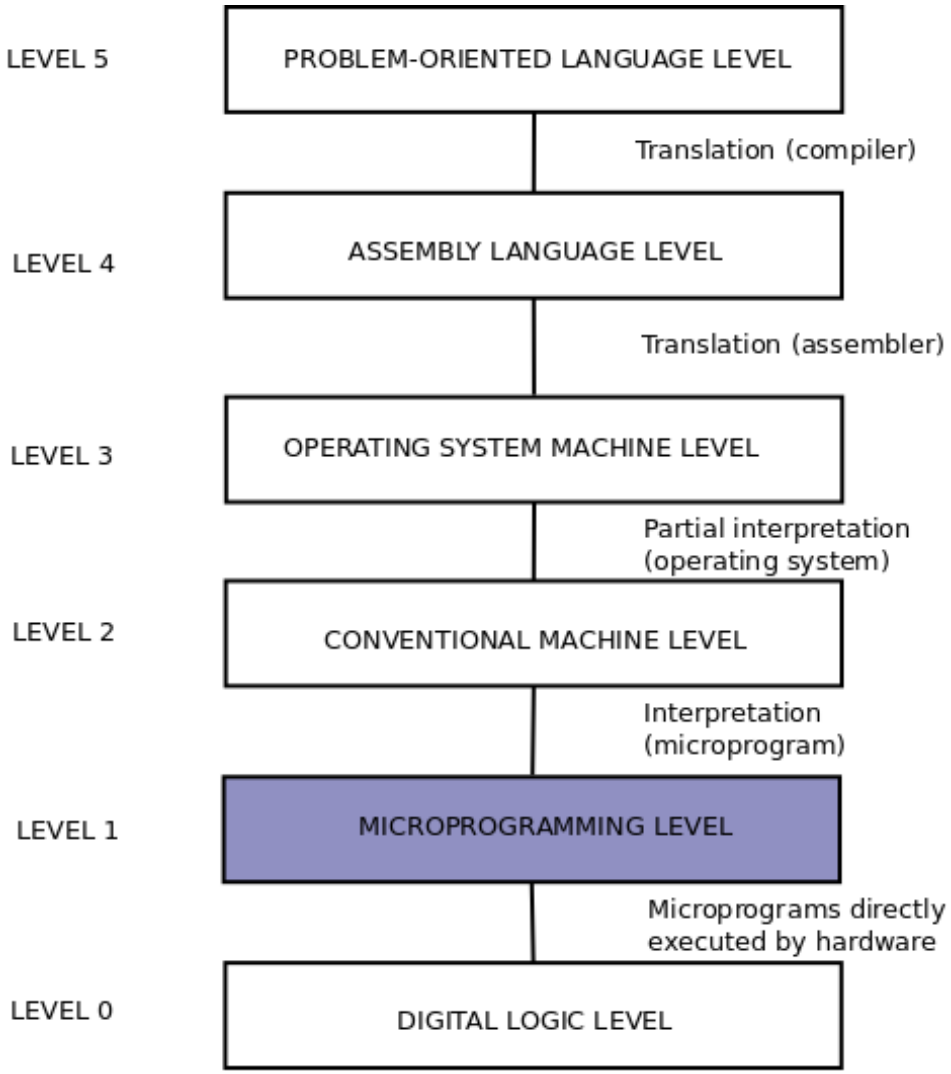
3rd International Conference on Computer Research and Development
Shanghai, March 2011

Overview

- MMP16 stands for 16-bit Micro-programmed Micro-Processor.
- MMP16 is a didactic micro-programmed micro-processor for those students that, having studied the basics of digital electronics, need to understand the principles of computer organization prior to study advanced computer architecture.
- MMP16 has been designed as a bridge between the study and use of digital electronic components and a full implementation of a digital microprocessor.
- These concepts have been identified as a teaching need by authors such as Hennessy, Patterson, Tomek and Tanenbaum.
- Besides the theoretical approach, MMP16 may be used as a learning tool, allowing the students to develop and test their own microprograms.

Motivation

Tanembaum defines the following levels, present on most modern computers (Tanenbaum, SCO, 2006).



Motivation

Tanembaum's model

- L2. Conventional machine level
- L1. Microprogramming level
- L0. Data logic level

Knowledge areas

- Computer architecture
- Principles of comp. organization
- Digital electronics

Main Design Features

- The main idea behind MMP16 is to build a bridge between Level-0 and Level-2 of Tanenbaum's model using simple circuits that any student with basic knowledge of digital electronics is able to understad.
- As a didactic tool, efficiency and performance are not the main design goals.
- By learning MMP16, students understand that they would be able to build a CPU following Von Neumann's model.

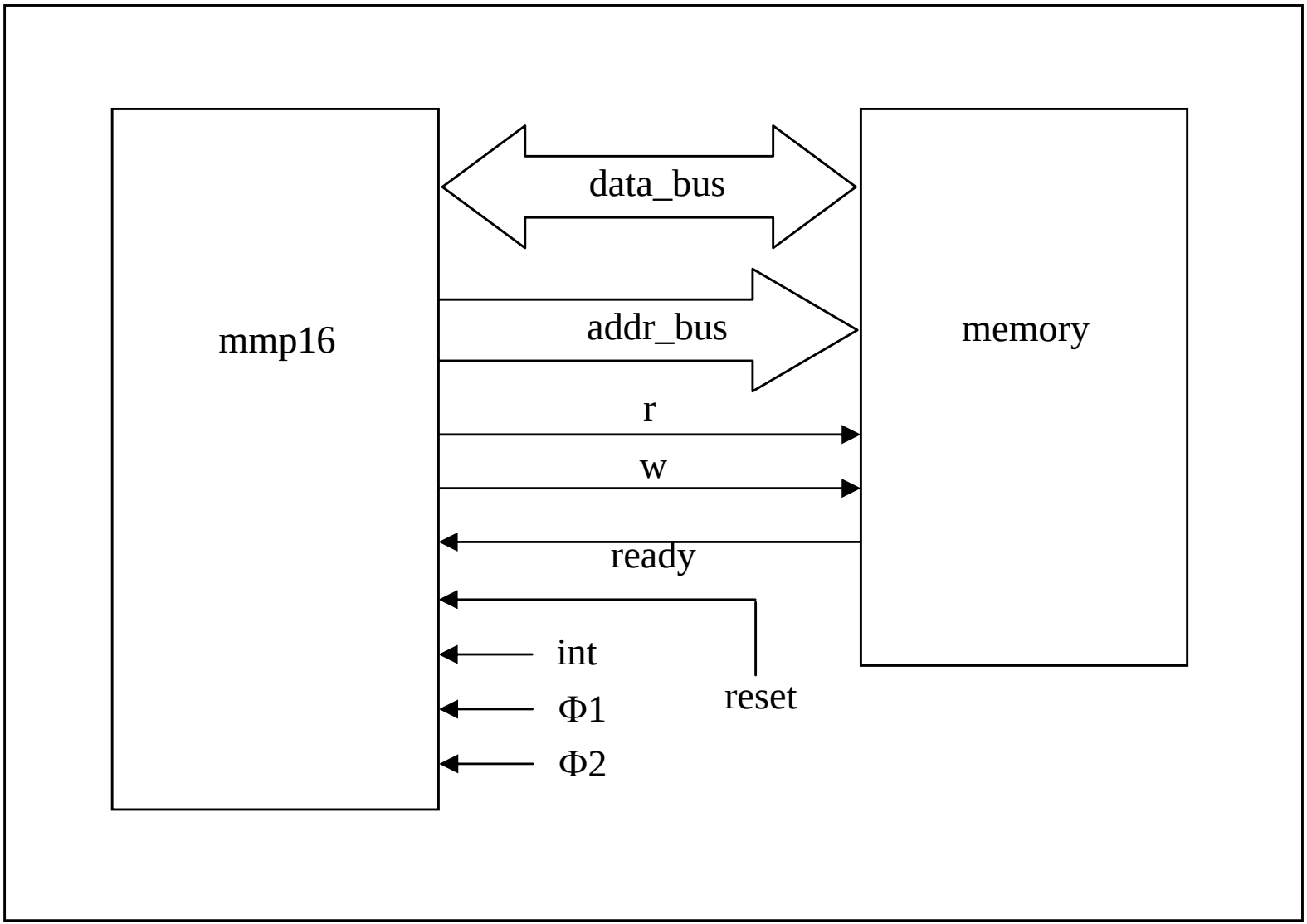
Main Design Features

- Why micro-programmed (and not wired)?
  - MMP16 is a didactic tool.
  - MMP16 is micro-programmed, and not wired beacuse of the simplicity of the former over the latter.
  - Allows the students to design automata corresponding to the instructions sequencing without having to struggle with the hardware, so they don't need to modify the software simulator.
  - Segmentation is easier to understand (extending segmentation to the data processing unit, as done in RISC superescalar machines, is easier after having understood this example).

Main Design Features

- Simplified architecture and reduced instruction set, but not RISC.
- Set of 16 16-bit registers, including SP and PC.
- 16-bit ALU able to perform arithmetic operations such as addition, substraction and logic ones such as AND, OR, XOR and NOT.
- 16-bit address and data bus.
- 64K 16-bit memory.
- Segmented control unit.
- No I/O devices. Design is focused on CPU.
- External signals are: r (memory read), w (memory write), ready (used by memory to inform the processor that the required operation has concluded), reset (resets CPU) and int (interrupts, may be activated in the software simulator.)

General Description



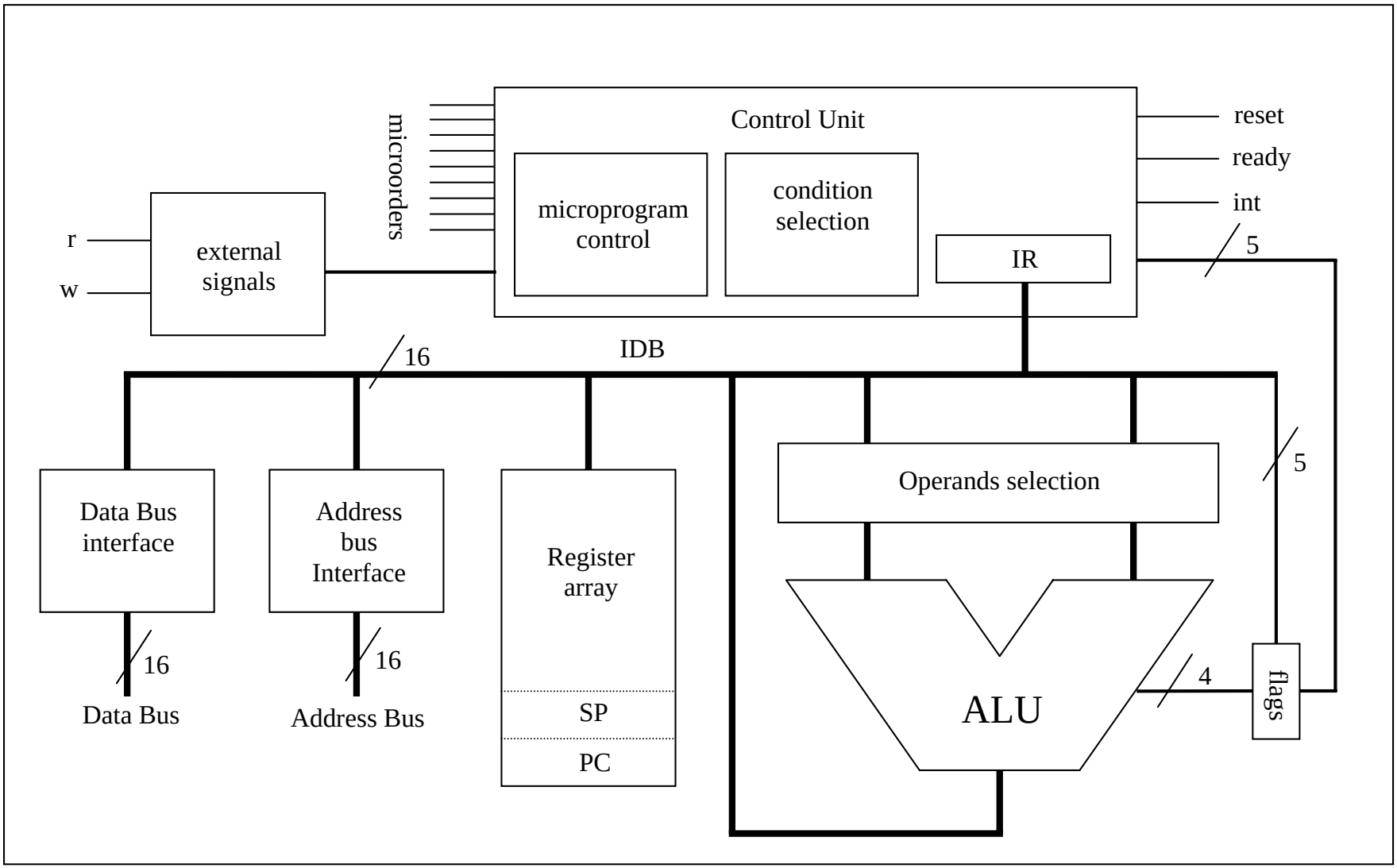
Organization

- A 16-bit Internal Data Bus (IDB) connects all the functional areas.
- Every register and bus are 16-bit ones, and a byte word cannot be addressed, in order to achieve the highest simplicity.
- Its control memory has 32-bit 256 words. 8 bits microaddresses are needed to address the control memory, each word contaning a microinstruction.
- The main goal of MMP16 is to consolidate the student's knowledge about CPU innner organization and instruction sequencing.

I/O Devices

- For the sake of simplicity, there are not I/O devices, but they could be easily added.
- Even though there are not separated I/O and memory maps, this is not a architectural limitation, used from the early PDP-11 to x86 architecture.
- Nevertheless, interrupts are already present in MMP16.

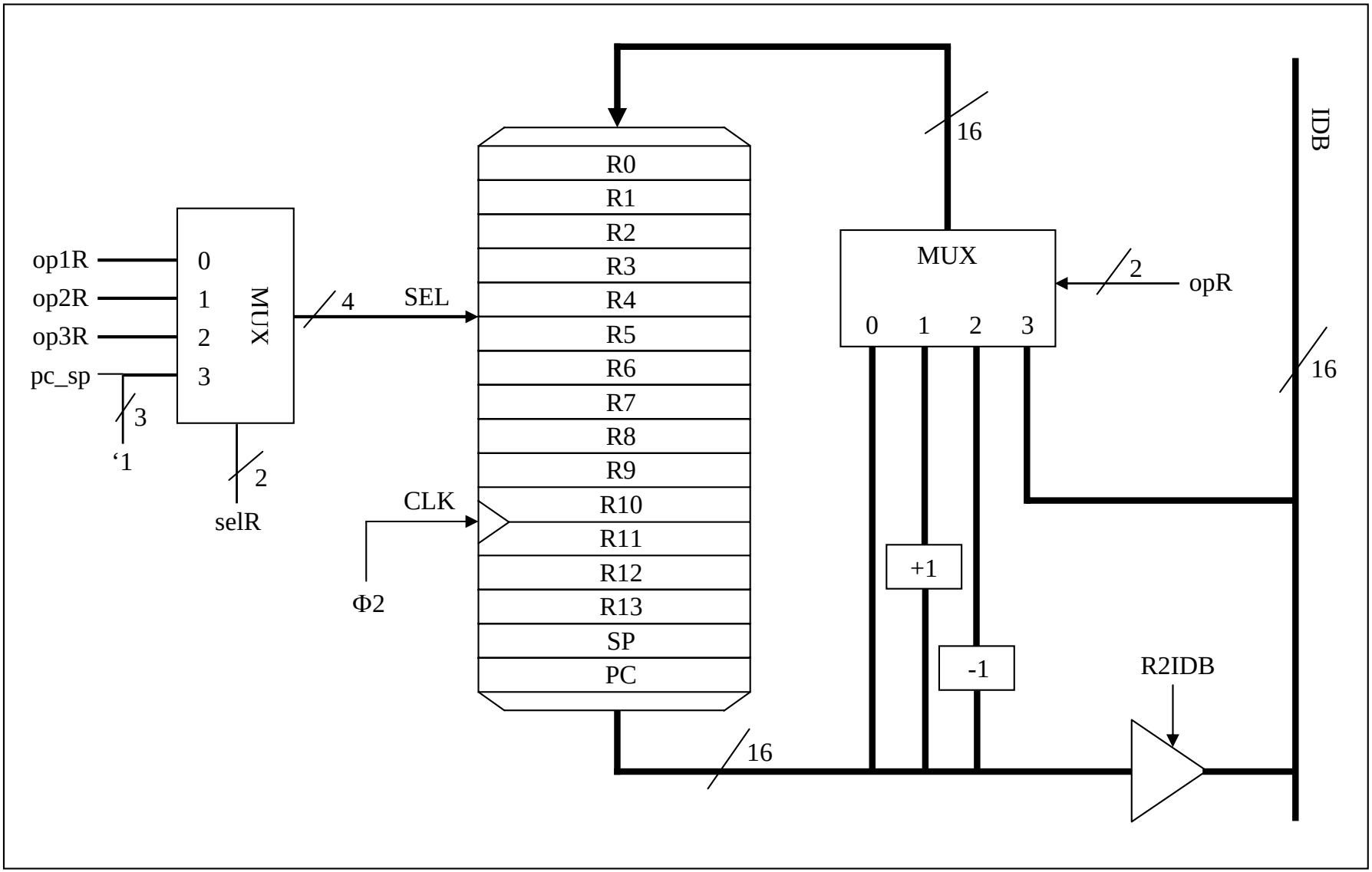
System layout



Register array

- There are 16 general purpose registers.
- R14 is the stack pointer (SP) and R15 is the program counter (PC).
- Registers may be addressed in two cases: a instruction referring a register is executed, or the control unit selects SP or PC for an operation to update PC or to handle the stack.
- To increment and decrement PC and SP faster, an incrementer and a decrementer are available, to avoid the use of the ALU for these operations.

Register array



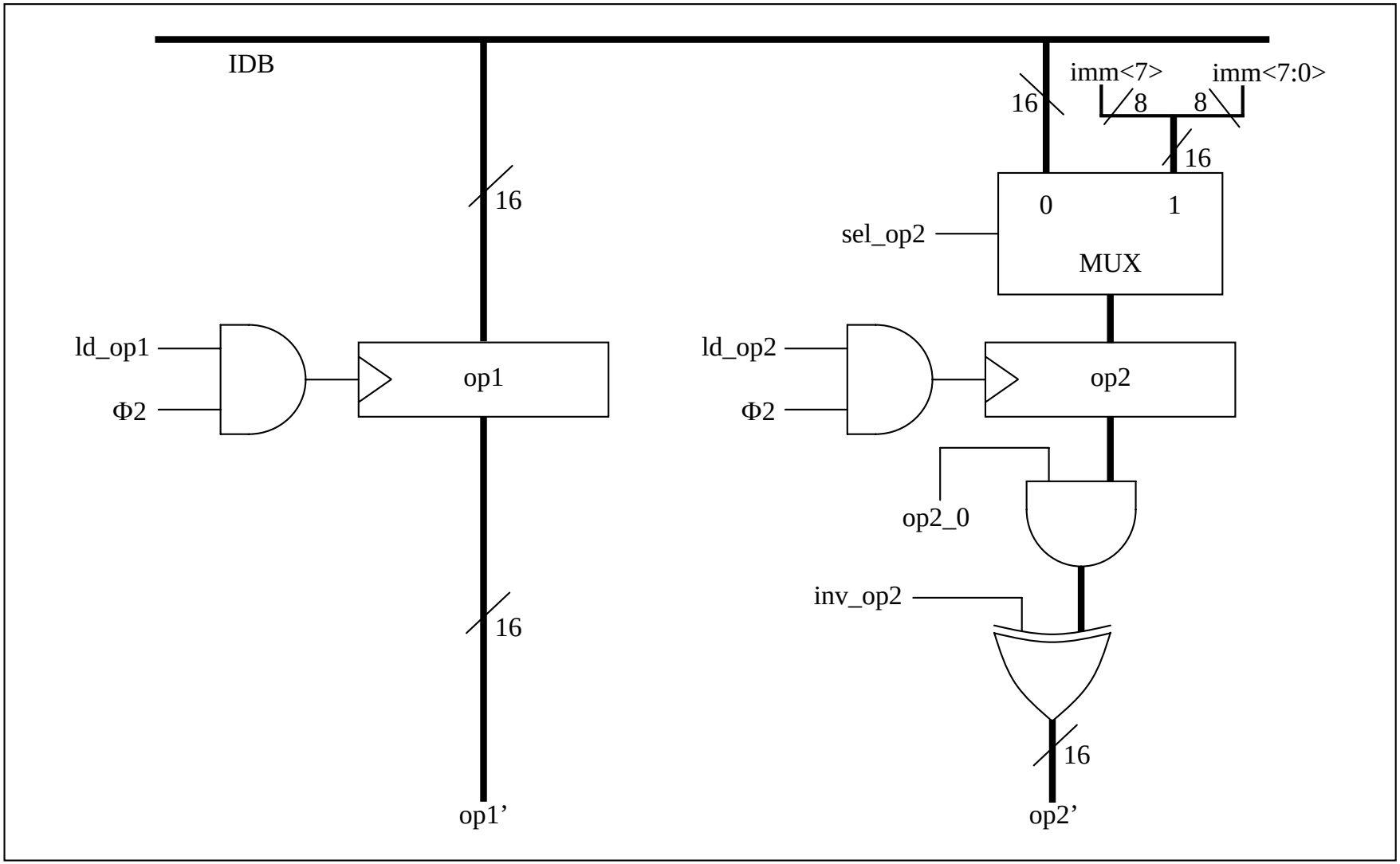
Register array

- seIR is a microinstruction field that selects how the register array is addressed: one of the two fields of the instruction register (IR), op1R (0), op2R (1) or op3R (2); or control unit directly selecting PC or SP.
- In this case, the control signal *pc_sp* allows selecting between PC and SP.
- opR is a microcommand used to select the operation to perform with the selected register: nothing (0), increment it (1), decrement it (2) or load it from IDB (3).
- Register array has a selection input (SEL) that selects the register whose data is obtained and that will be updated.
- Finally, the control signal R2IDB lets the content of the selected register to go IDB through a tristate. RaIDB is obtained decoding aIDB command from the microinstruction.

Operand selection

- The first ALU operand (op1) can be loaded from IDB using the control signal *ld_op*1.
- *Sel_op*2 selects if the second operand (op2) is loaded from IDB or is loaded with the 8-bit immediate field of IR (sign extended)
- *Ld_op*2 allows to load this second operand with the selected value.
- ALU's first operand will always be op1, but the second one may be modified to perform several special operations, such as subtractions using "two's complement" representation for negative numbers.
- It is possible to force op2 to have all its bits to one or to zero.

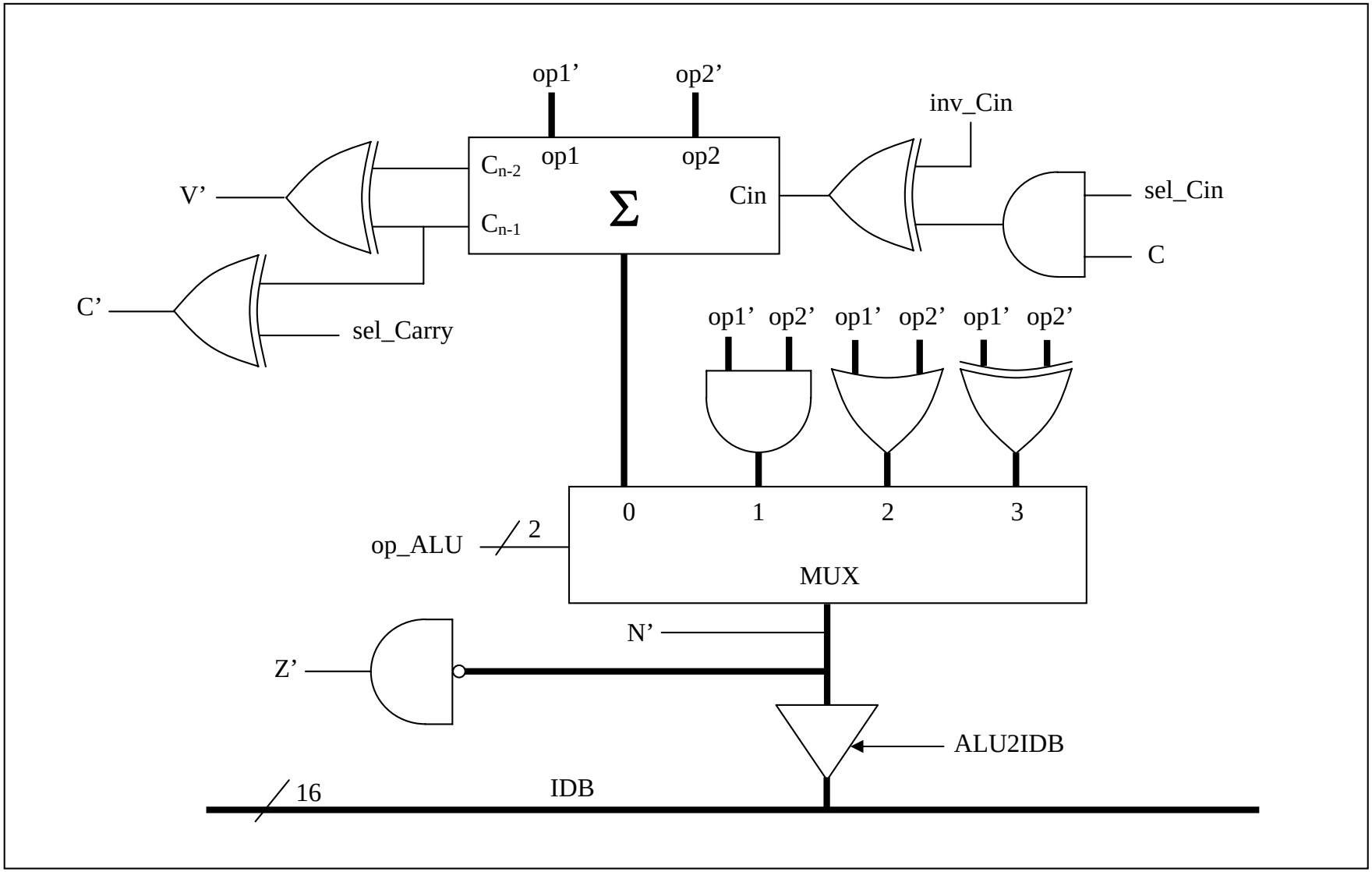
Operand selection



Arithmetic Logic Unit (ALU)

- The 16-bit ALU is able to perform arithmetic operations such as addition, substraction (using "two's complement") and logic ones such as AND, OR, XOR and NOT.
- The ALU includes an adder and three sets of logical gates.
- The NOT operation has only one operand (op1) and it is performed through a XOR operation with 1...1 as a second operand.
- The operation to be performed in the ALU (addition, substraction, AND, OR, XOR, NOT) can be selected using op_ALU.
- The signal ALUaIDB, obtained decoding aIDB microcommand, allows the result of the ALU to go to IDB through the appropriate tristate.

Arithmetic Logic Unit (ALU)



Arithmetic Logic Unit (ALU)

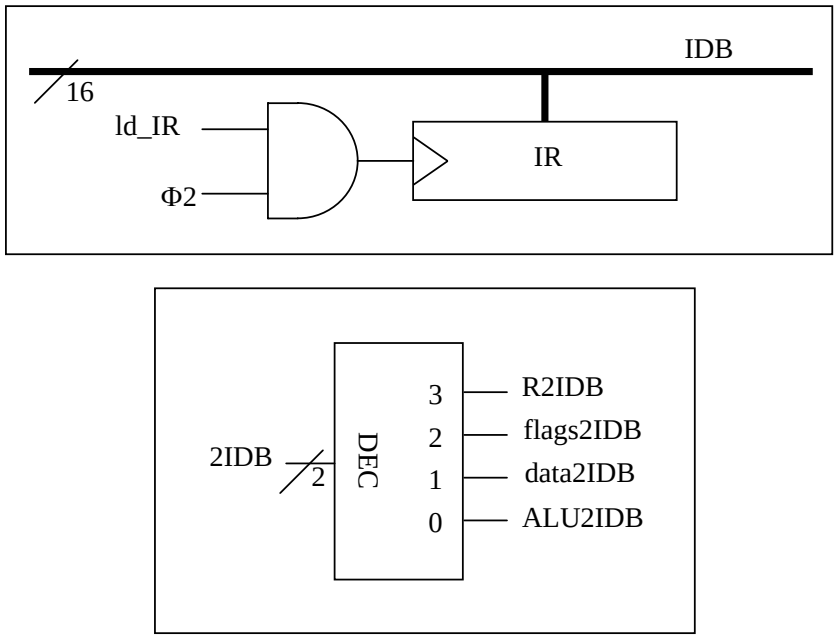
- When performing an addition or substration, initial carry has to be set using the sel_Cin and inv_Cin signals.
  - Initial carry is 0 for a addition.
  - Substration is performed adding op1 and op2's two's complement (negated op2 and initial carry 1).
  - To perform a 32-bit addition, both 16 LSB are added with initial carry 0 and then both 16 MSB using as initial carry the one generated in the LSB addition.
  - To perform a 32-bit substraction, both 16 LSB are substracted using borrow as carry flag, and then both 16 MSB using as initial carry the negation of the one generated in the LSB substraction.

Flags

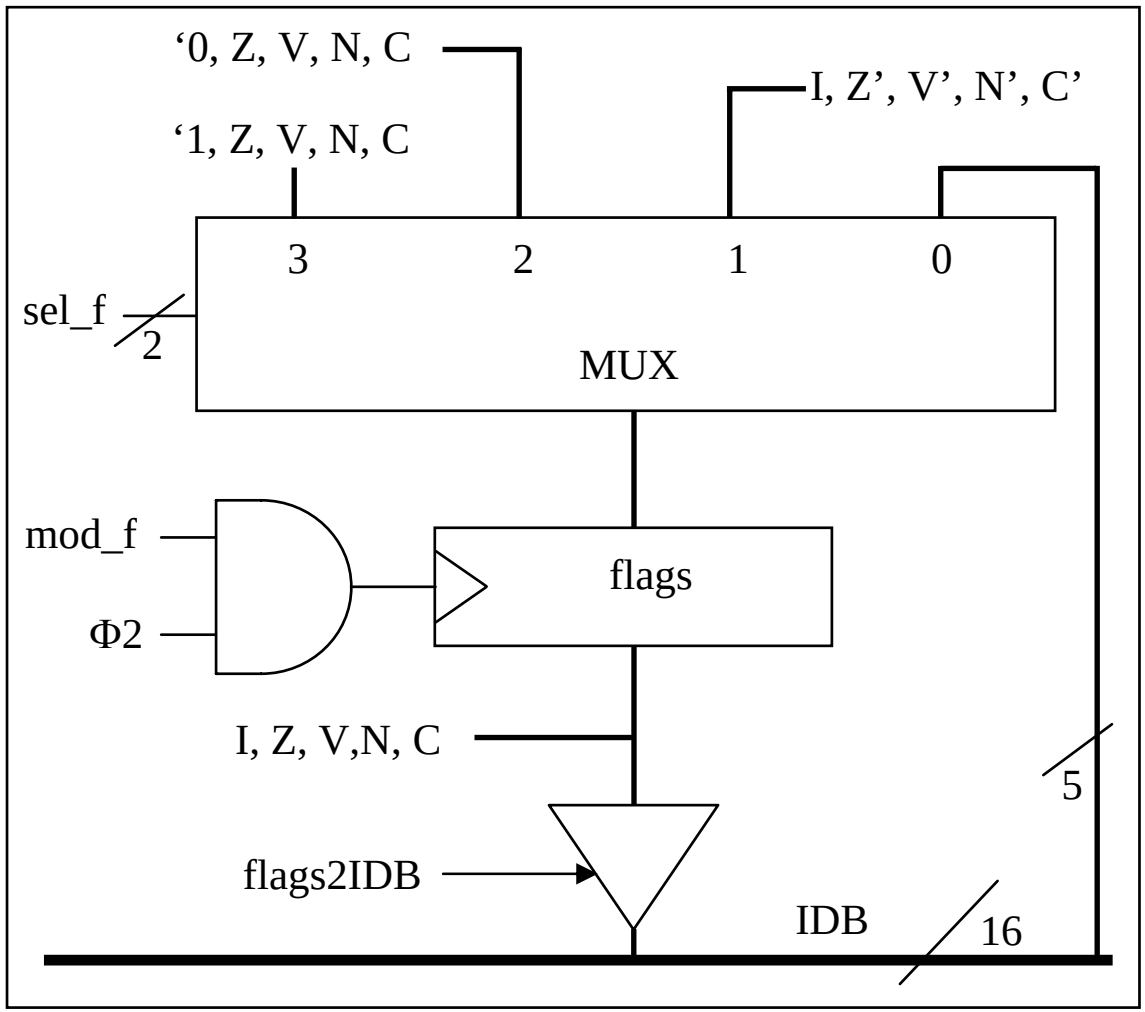
- Flags indicate four conditions that can occur as a result of a operation in the ALU, as well as one bit indicating if interruptions are enabled or disabled.
- The four possible conditions are:
  - Z: 1 if result is zero and 0 otherwise.
  - N: 1 if result is negative (MSB equals 1), and 0 otherwise.
  - V: 1 if overflow in a two's complement operation is detected.
  - C: Cn-1 en case of addition (carry) and NOT Cn-1 in case of substraction (borrow).
- To indicate interruptions:
  - I: 1 if interruptions are enabled, 0 otherwise.

Flags

- Sel_f allows to select if flags are loaded from the data bus, if they are updated from flags generated in the ALU, if interruptions are enabled or not. Mod_f allows to update the flags register.
- The flagsaIDB signal activates the tristate that allows the flags to go to IDB (i.e. go to stack). This signal is obtained decoding aIDB.
- Flags register is 5-bit, so they will be the five LSB from IDB.



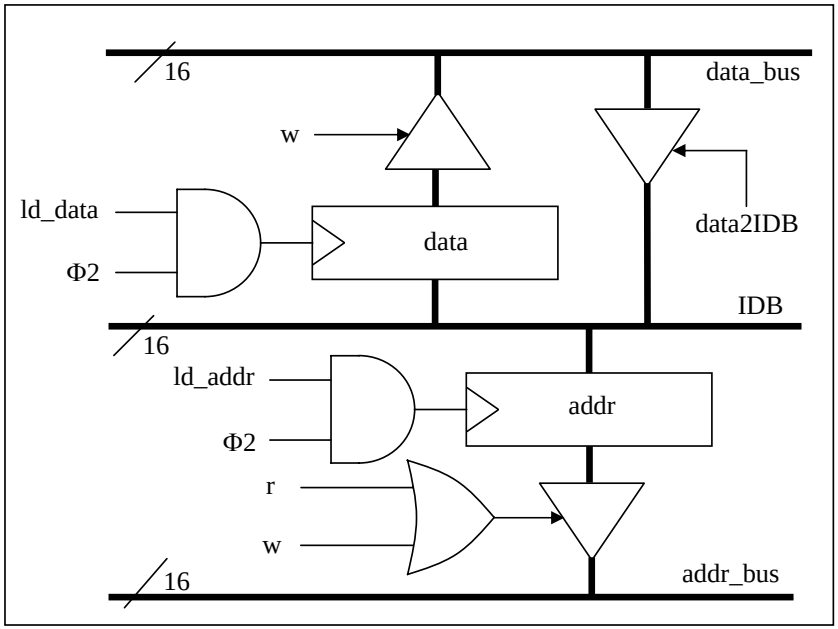
Flags


‘0, Z, V, N, C
‘1, Z, V, N, C
I, Z’, V’, N’, C’
3
2
1
0
sel_f
2
MUX
mod_f
Φ2
flags
I, Z, V,N, C
5
flags2IDB
IDB
16

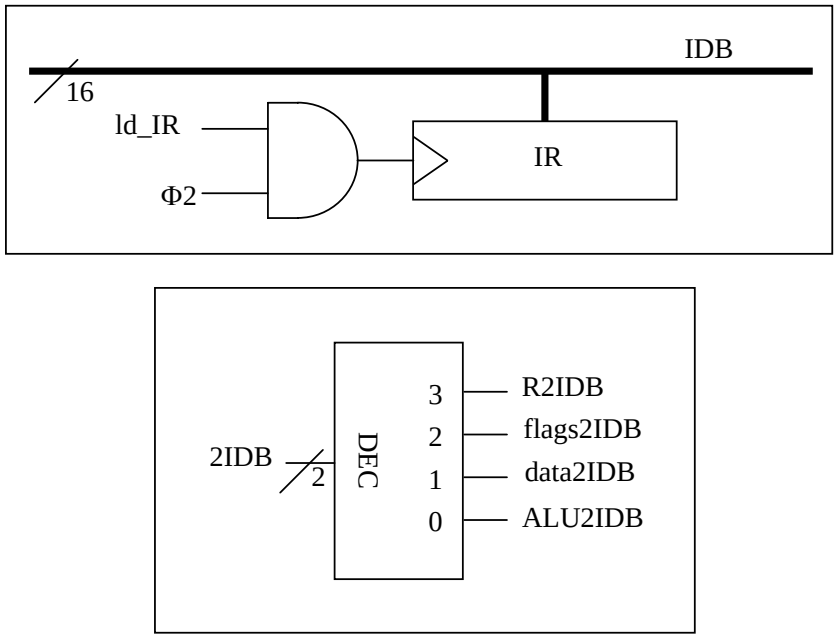
Interfaces with external buses

- Ld_data allows to load the data register from IDB.
- When the w signal is activated, data from this register will go to the data bus so they can be written on memory.
- Data register keeps the data stable in the data bus during all the memory write cycle.
- Data can go from the external data bus to IDB through a tristate control by dataaIDB generated from aIDB.
- Ld_addr enables loading the addr register from IDB. When the signales r or w are activated, its content will go to the address bus.
- Addr register keeps the address stable durgin both memory read or write cycles.
- IR can be loaded from IDB activating the ld_IR signal.

Instruction Register and aIDB decoding (I)



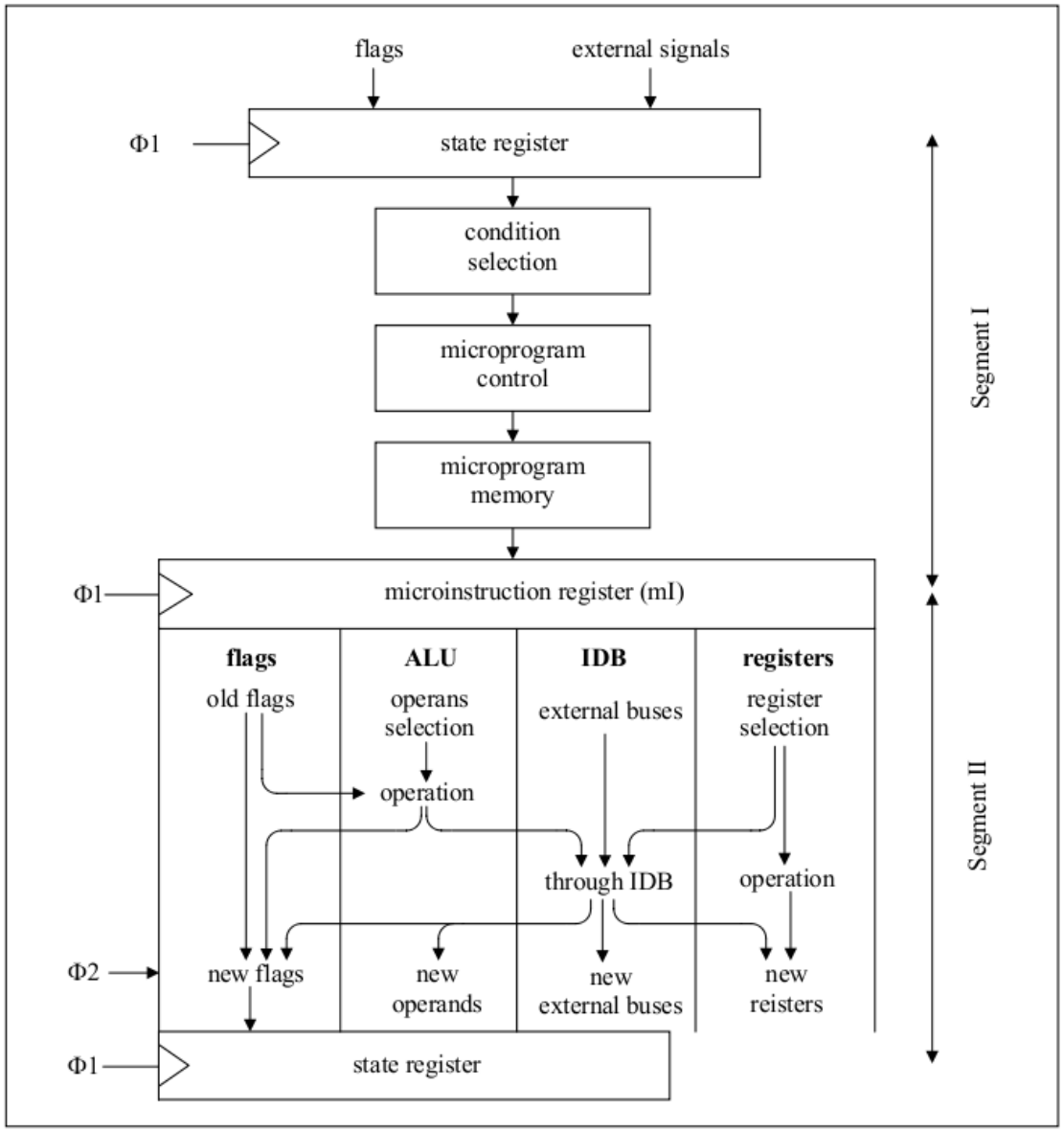
Instruction Register and aIDB decoding (II)



Pipelining

- The control unit of MMP16 is segmented:
  - Segment I is devoted to the search of the next microinstruction to execute.
  - Segment II is devoted to the execution of the current microinstruction.

Pipelining


flags
external signals
Φ1
state register
condition selection
microprogram control
microprogram memory
Segment I
Φ1
microinstruction register (mI)
flags
ALU
IDB
registers
old flags
operans selection
external buses
register selection
operation
through IDB
operation
Φ2
new flags
new operands
new external buses
new reisters
Segment II
Φ1
state register

Pipelining

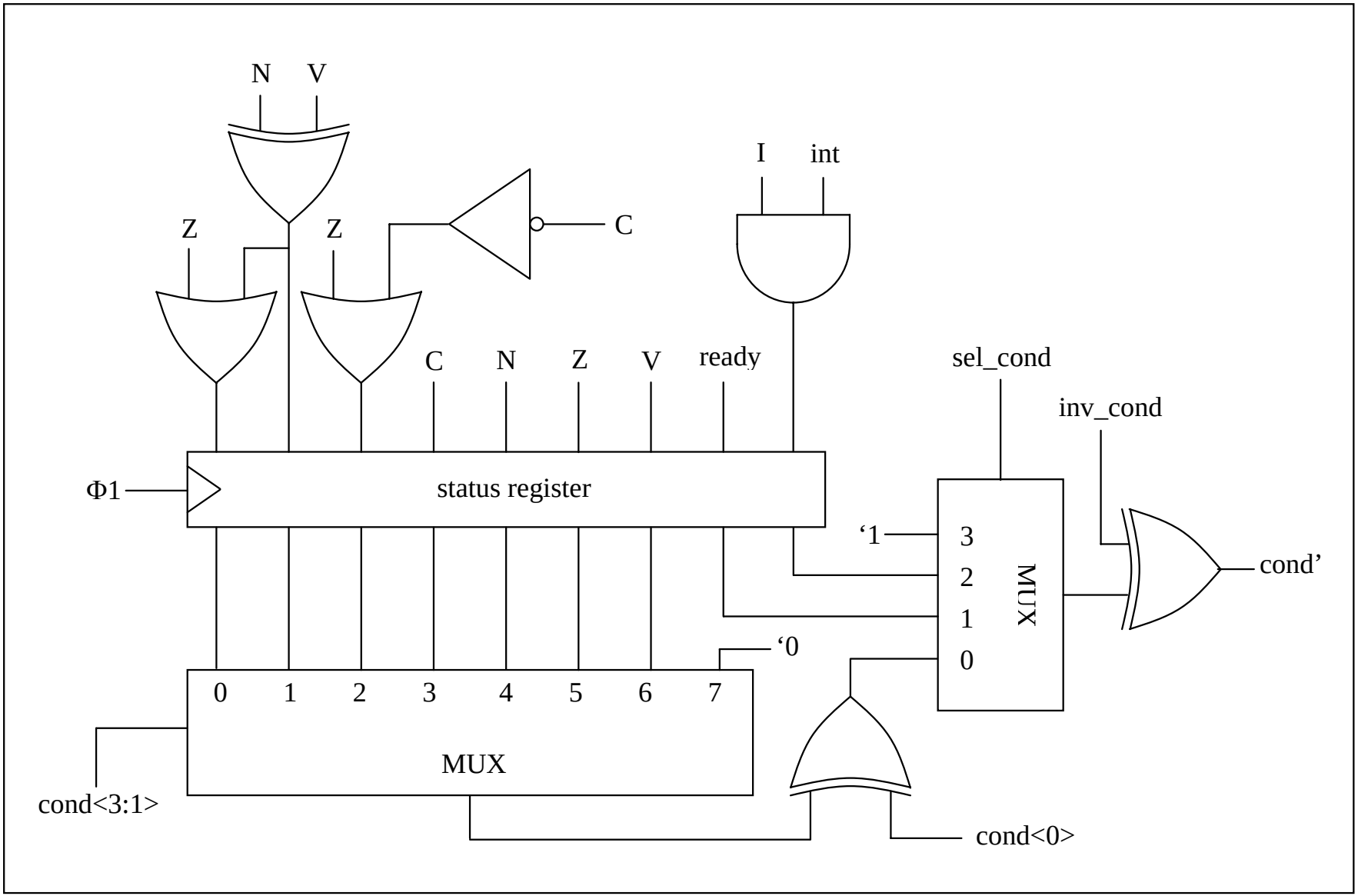
- This kind of pipeline is one of the most frequent ones. It tries to parallelize two slow processes: the search of a new microinstruction in the memory and the execution of the microinstruction, and therefore accelerate the machine.
- The basic feature of this kind of organization is that the election of next microinstruction in the case of a conditional jump inside the microprogram is performed depending on the status of the machine at the end of the previous microinstruction, and not on the status of the machine at the end of the current cycle.

$$maddress_{i+1} = f(minstruction_i, status_{i-1})$$

Condition selection

- The status register stores the conditions in function of which is it possible to modify the ordinary program execution sequence.
- Some of these conditions are:
  - Those indicated in the "cond" field of IR, based on values of the flag registry.
  - The "ready" condition used for wait cycles until memory operations are completed.
  - When interruptions are allowed (flag I), activating the interruption request (int) external signal.

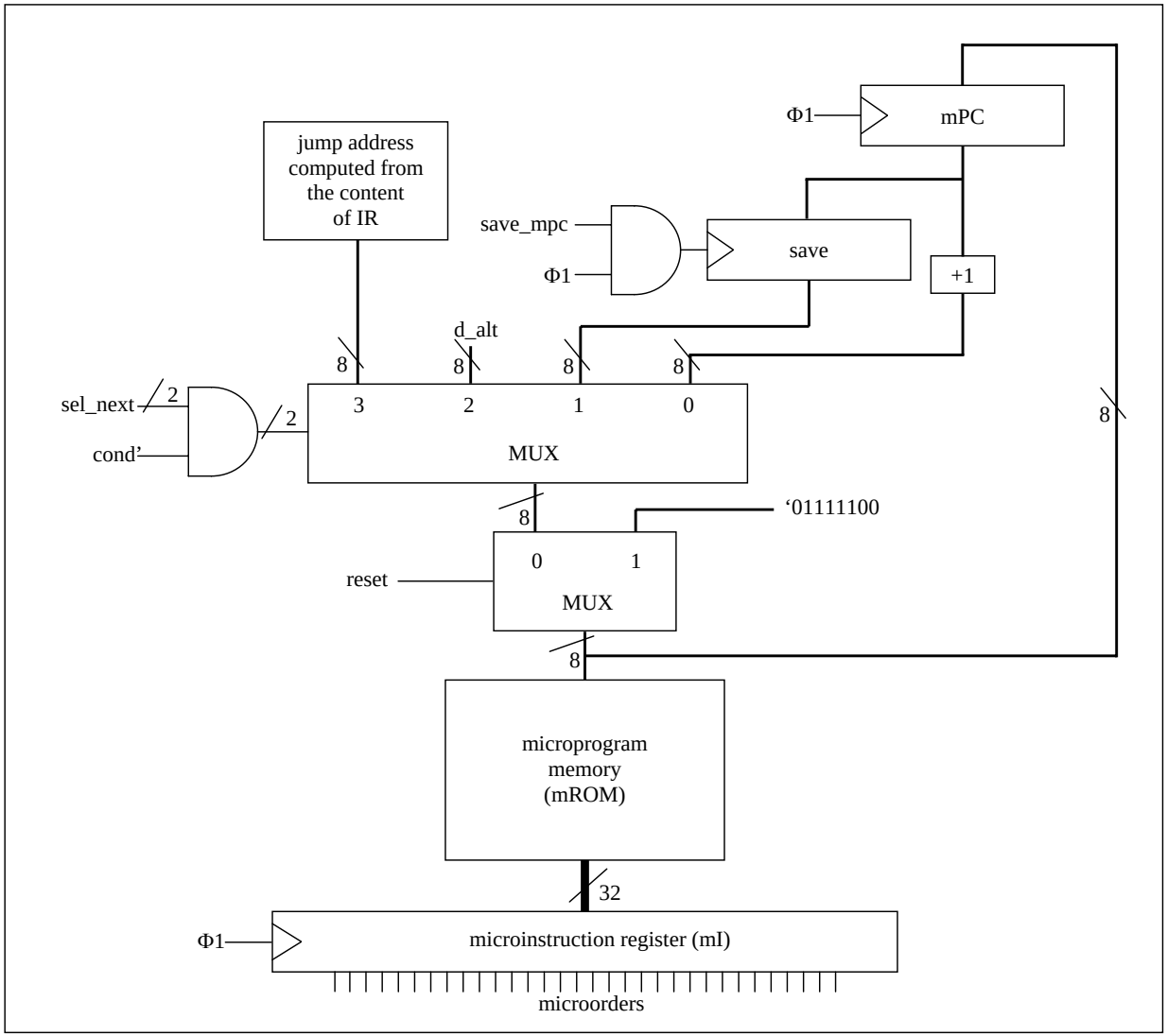
Condition selection



Microprogram control

- mPC is the microprogram counter, contains the control memory address where the microinstruction that is being executing is contained and that would be stored in the microinstruction register (mI).
- The selection of the next microinstruction (where am I going) is made depending on the selected condition (cond') and sel_sig, that indicate where am I going if selected conditions occurs. In case this doesn't occur, I will go to the next microinstruction, following mPC (mPC+1).
- Due to the segmentation of the control unit, jump condition that is evaluated in a given status is the one generated in the previous one.
- MMP16 has a external *reset* signal that allows to intialize the machine.

Microprogram control



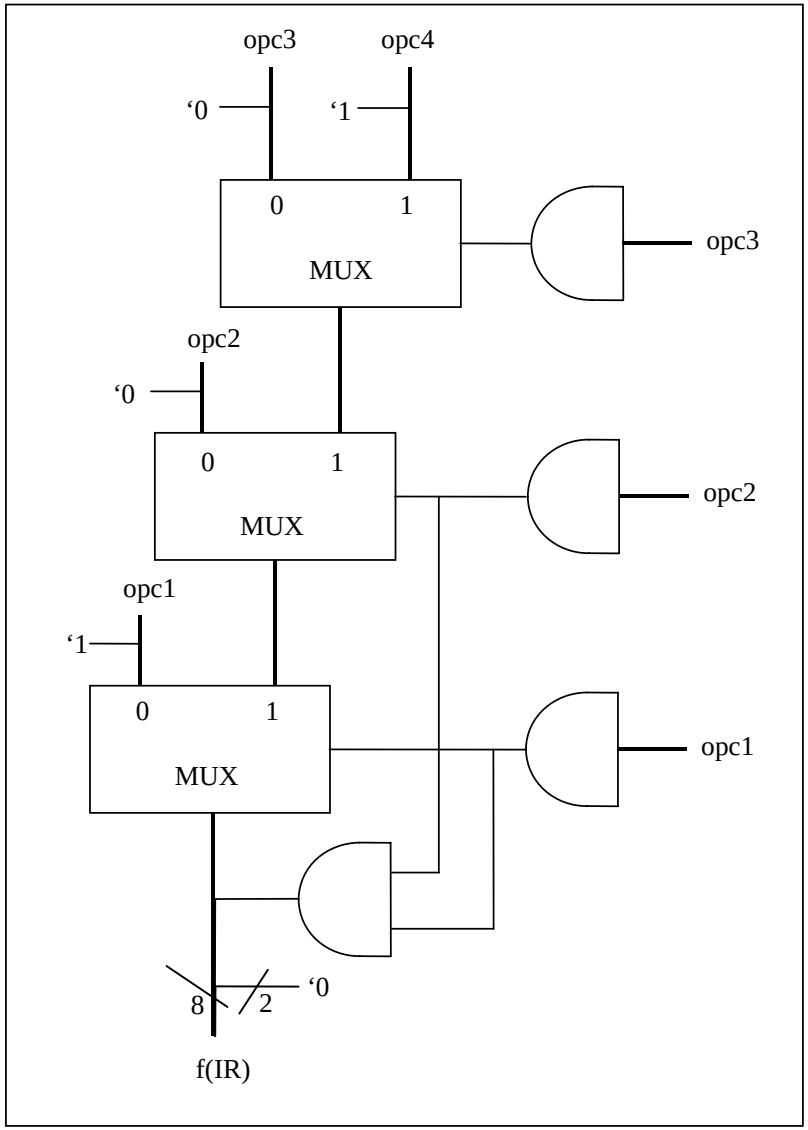
Microprogram control

- The "save" registry may be used to save any frequently used microaddress. It is loaded with the save_mpc signal.
- D_alt is a field of the microinstruction that allows to jump to any address of the control memory. As it overlaps with the microcommands that control ALU, if d_alt is used, ALU cannot be used in this status.
- It is possible to jump to any address depending on the instruction contained in the IR. The microprogram implementing this instruction should be loaded in this address. This is a basic possibility in any instruction interpreter.

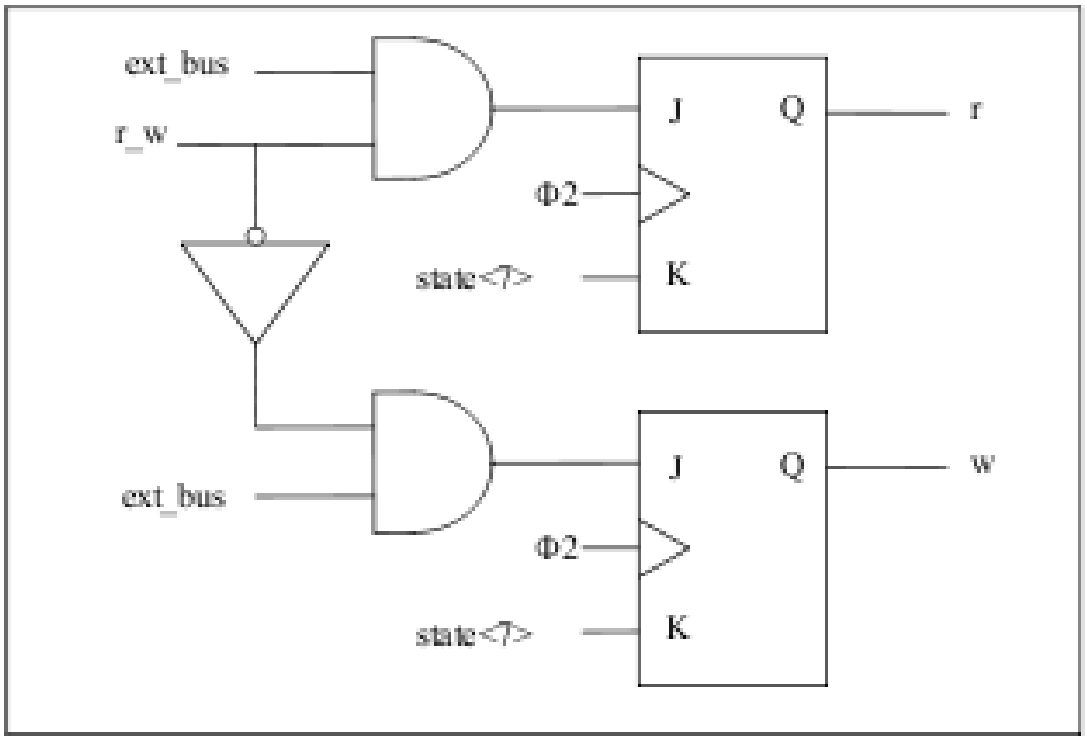
Decoding of the instruction

- The hardware of MMP16 determines the memory address where the microprogram the executes the corresponding instruction is located, what is commonly referred to as the decoding of the instruction.
- MMP16 operation codes may be 4, 8, 12 or 16-bit.
  - 16-bit operation codes have their 12 MSB (cop1, cop2, cop3) to 1.
  - 12-bit operation codes have their 8 MSB (cop1, cop2) to 1.
  - 8-bit operation codes have their 4 MSB (cop1) to 1.
- This allows 15 4-bit, 15 8-bit, 15 12-bit and 16 16-bit codes, up to 61 in total.

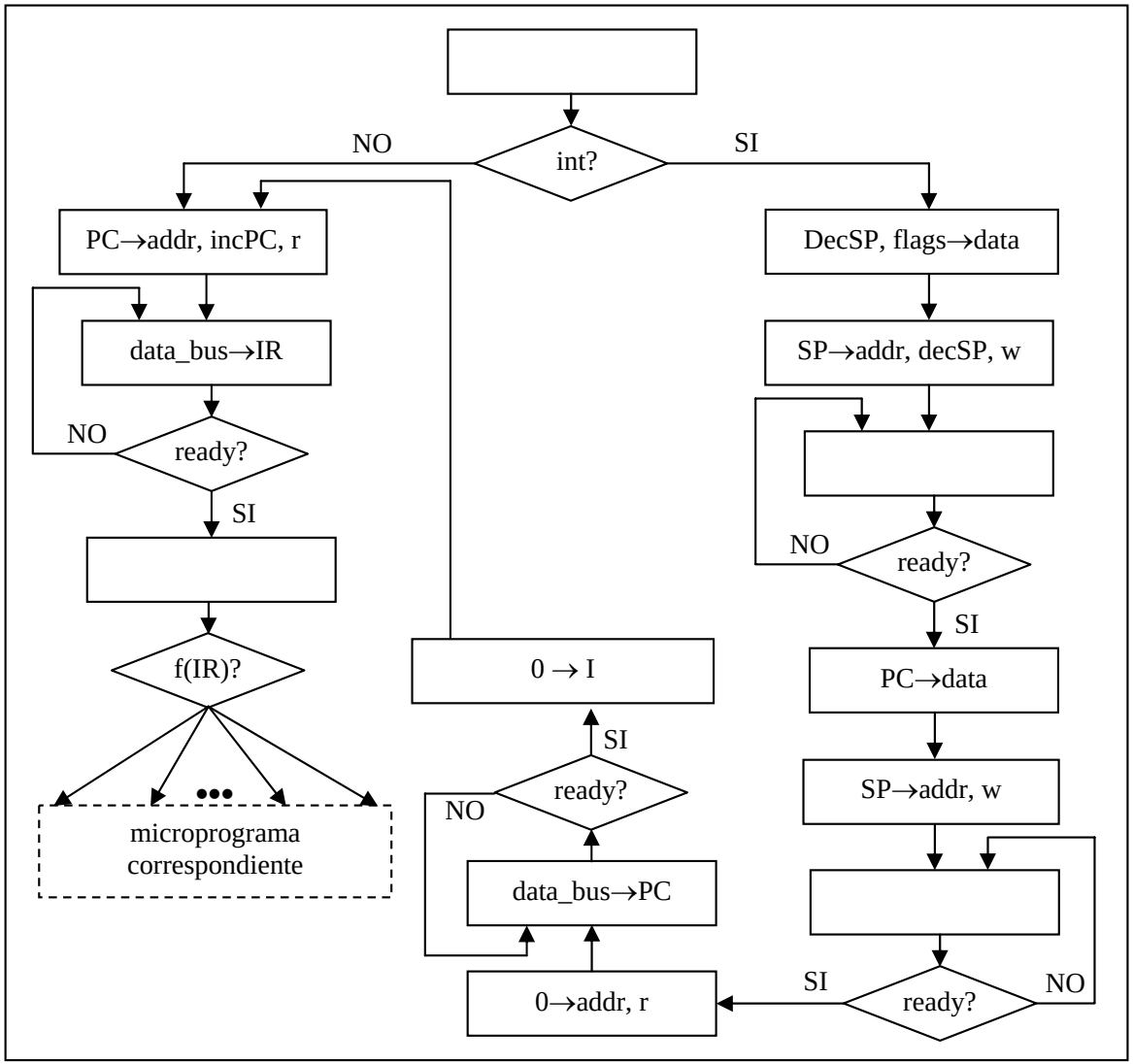
Jump address depending on IR



External signals generation



Fetch phase flux diagram



Software simulator

- MMP16 software simulator (available in both GNU/Linux and Windows platforms) is one of the project's advantages.
- The simulator helps to understand instruction sequencing and the data path management, and helps the students to comprehend the inner operation of the processor they have already studied in theory.
- The simulator is the last step in the process of connecting the student's knowledge about digital electronics (Tanenbaum's Level-0) to the principles of computer organization and the basics of computer architecture (Tanenbaum Level-2).

Software simulator

- A typical session would include the following steps:
  - Load the microprograms into the control memory.
  - Load the program into main memory.
  - Send a reset pulse.
  - Run the processor some clock cycles.
  - Test the content of the desired registers or memory words.

Conclusions

- As a didactic project, MMP16 covers the gap between digital electronics and computer architecture offering a complete set of tools:
  - The design of a modern computer.
  - The practical application of the principles of computer organization including advanced concepts such as pipelining.
  - The possibility to code and test microprograms.
  - A GNU/Linux and Windows-based software simulator as the last step of the process.
- Full information and downloads in: http://www.diatel.upm.es/jllopez/mmp16/