11: Σχεδίαση και Προσομοίωση Ολοκληρωμένων Συστημάτων με Verilog (III)

Ο Μικροεπεξεργαστής MicroCPU

Vasileios Tenentes

University of Ioannina

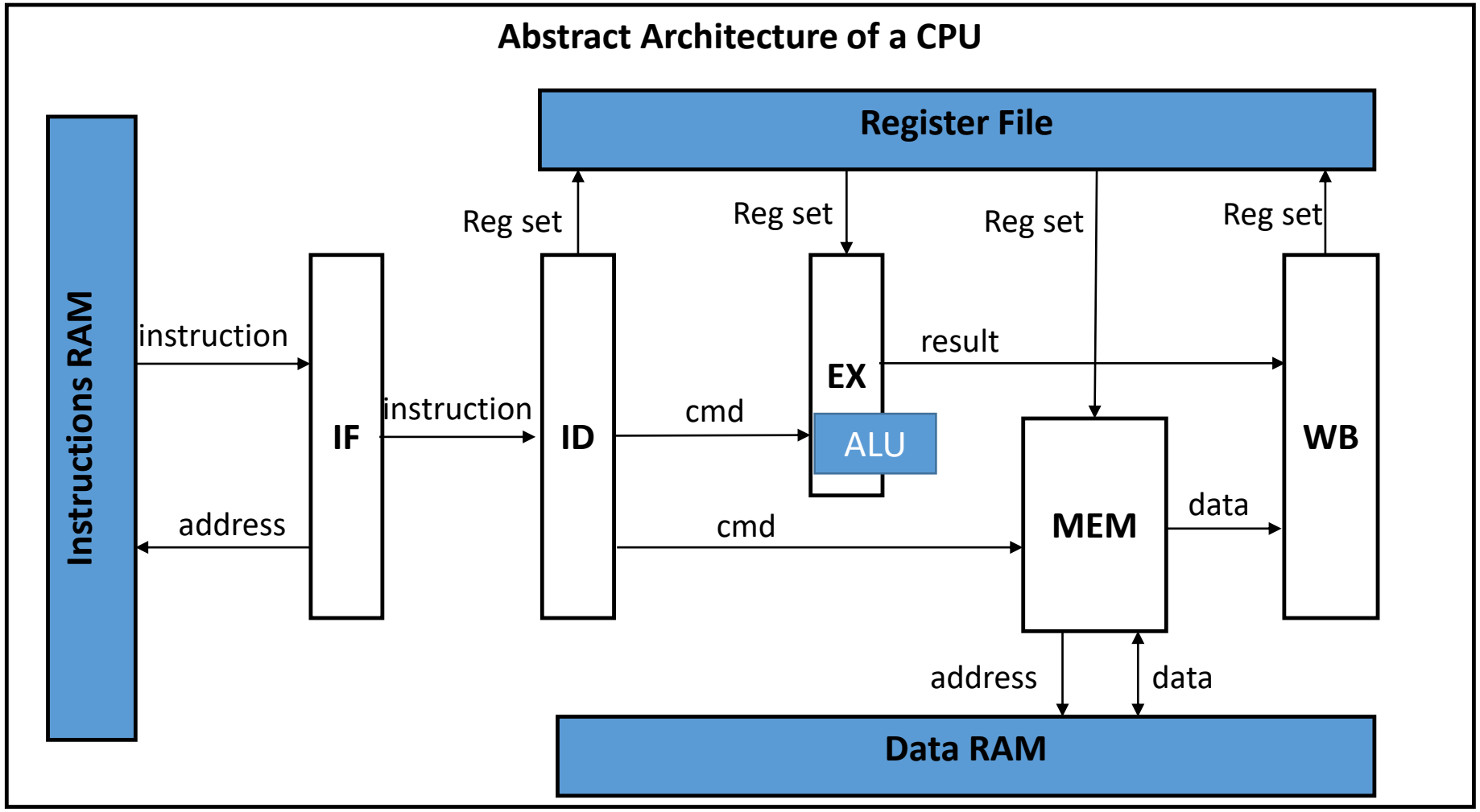
Outline

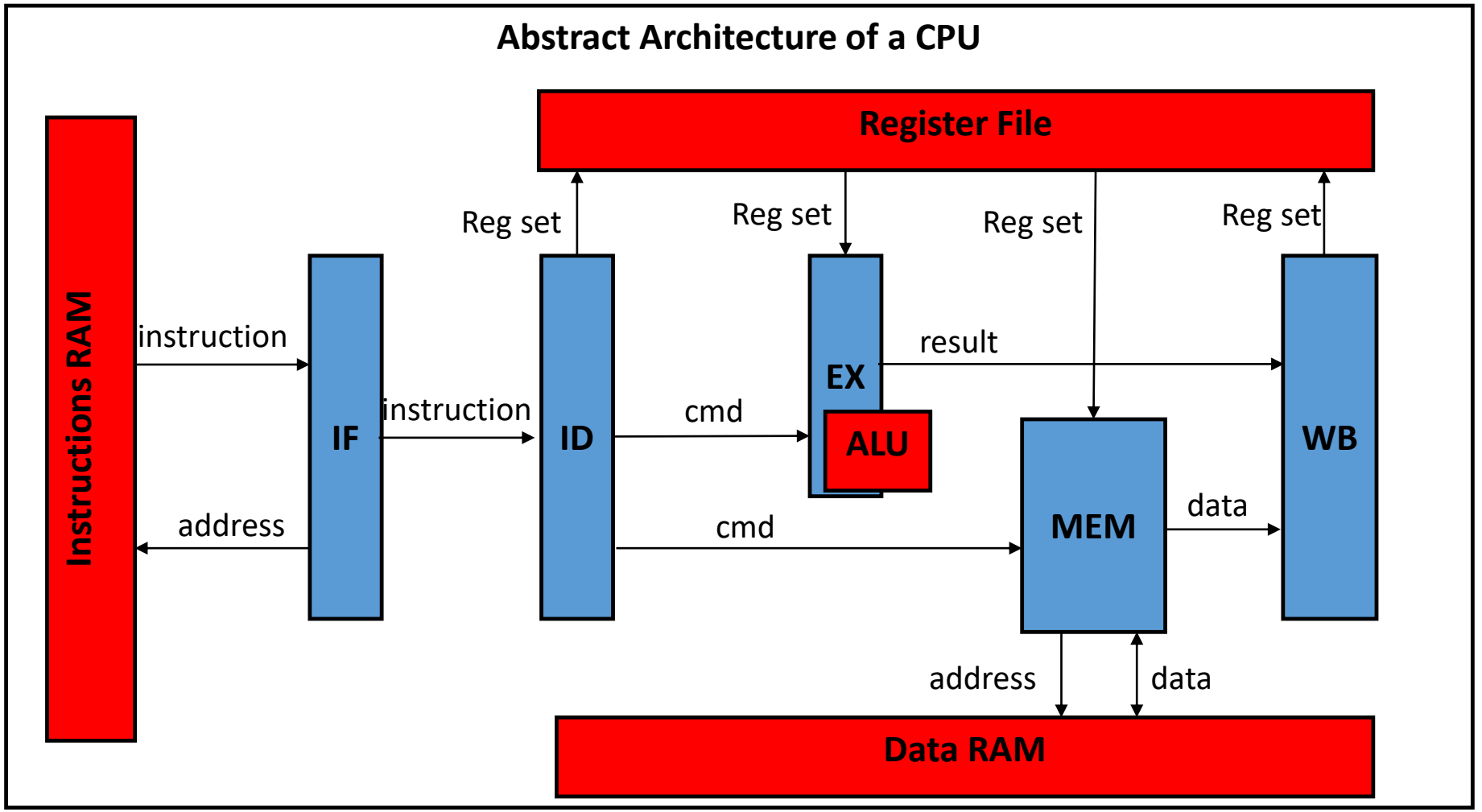
Βασική Αρχιτεκτονική Επεξεργαστών
Pipelining

Σχεδίαση σε Verilog επεξεργαστή

- **Μοντελοποίηση Μνήμης και Σχεδίαση του Ελεγκτή Μνήμης**
- **Σχεδίαση Αριθμητικής Λογικής Μονάδας**
- **Σχεδίαση Πεπερασμένου Αυτόματου Μονάδας Ελέγχου**

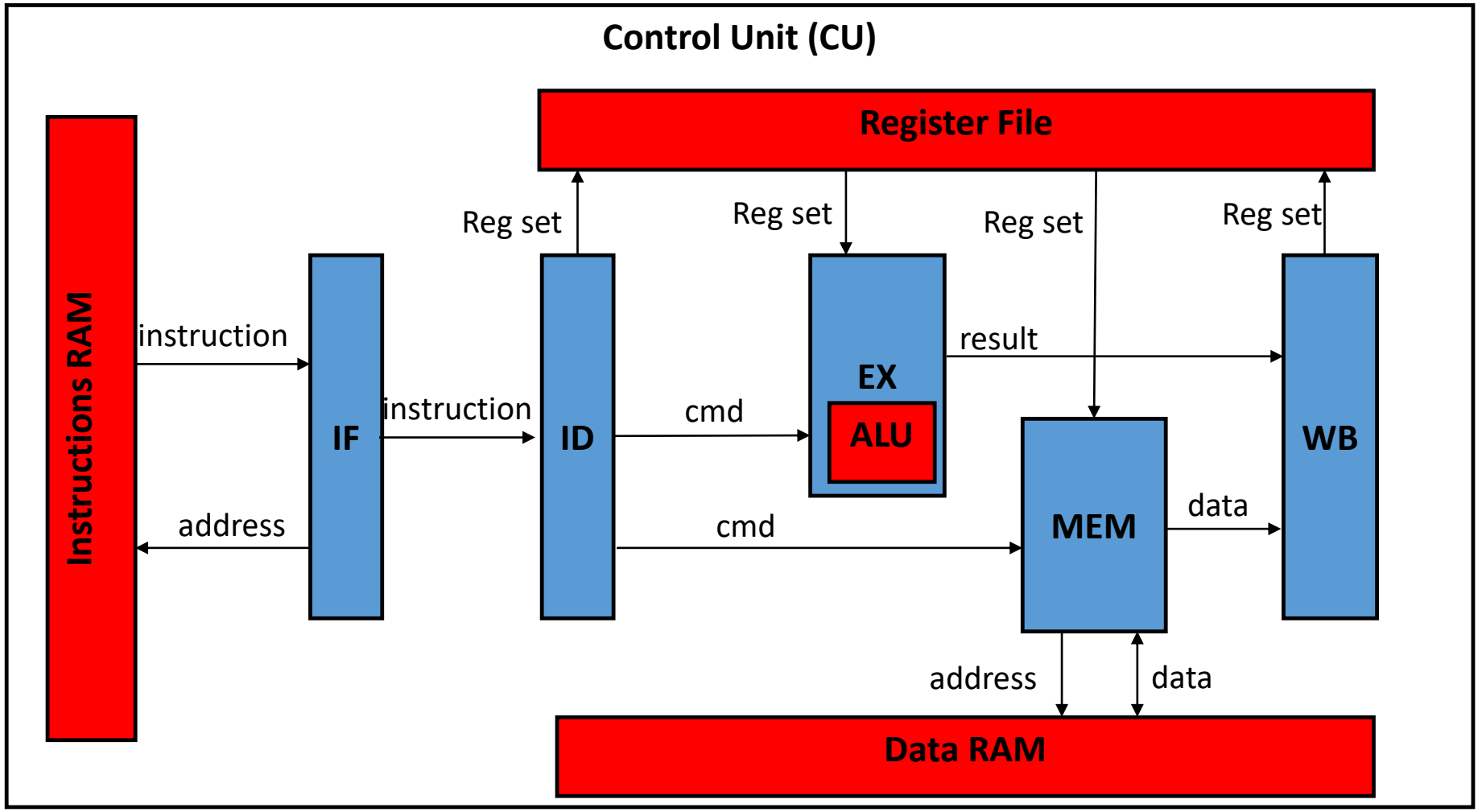
Βασική Αρχιτεκτονική Διοχέτευσης RISC Μικροεπεξεργαστών (RISC pipelining)





Components του επεξεργαστή

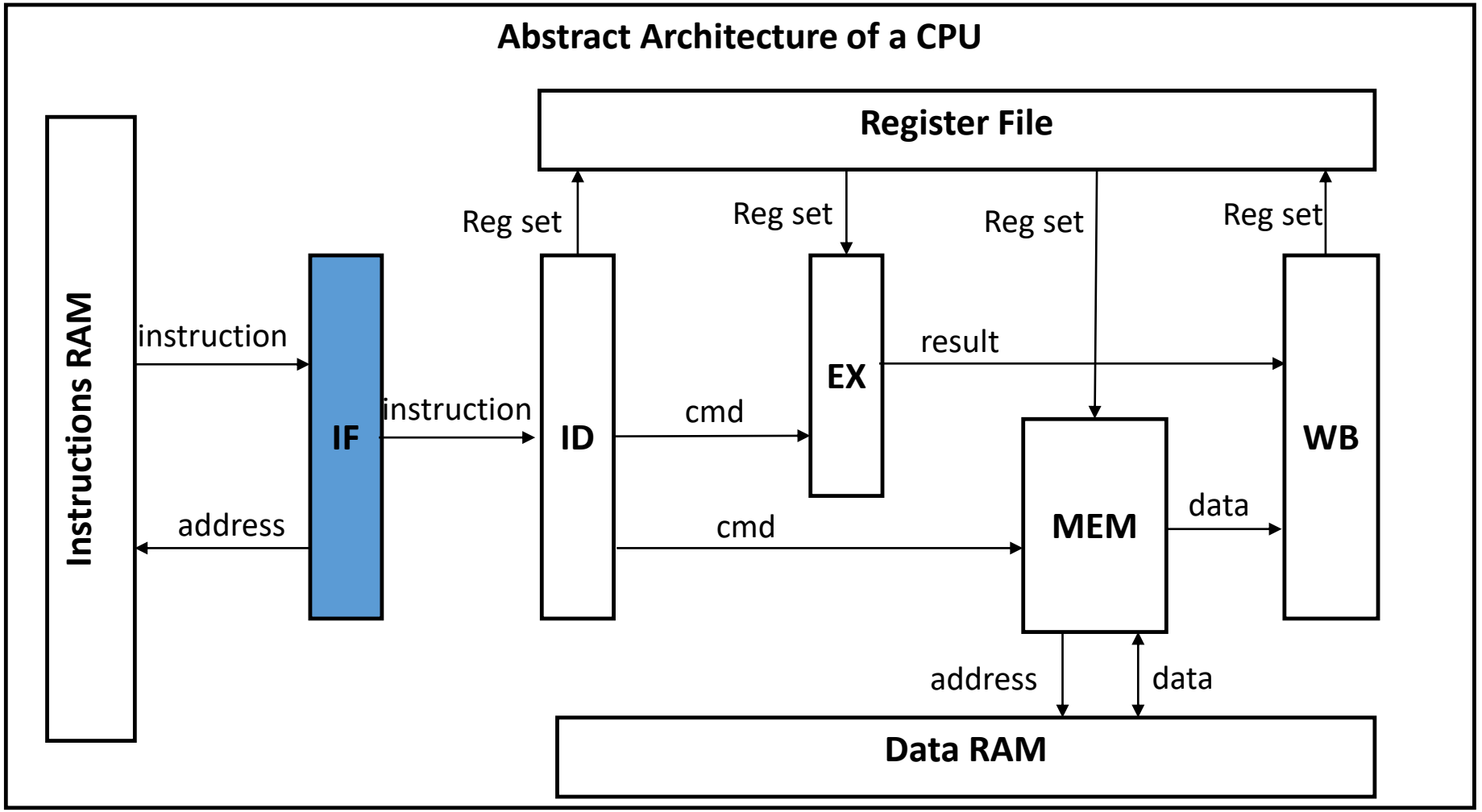
Πράξεις που εκτελούνται. Κυρίως από λογική της Control Unit



Components του επεξεργαστή

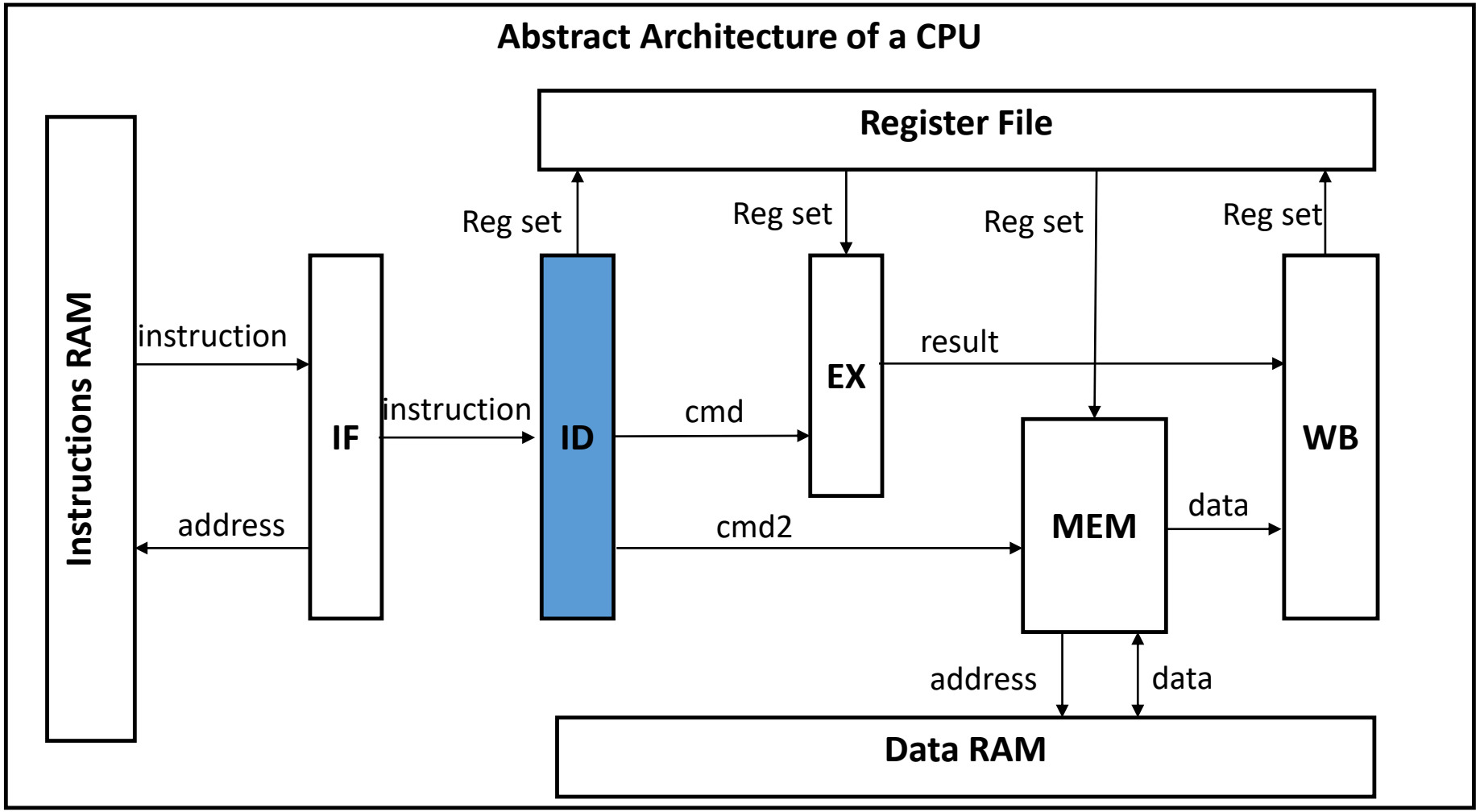
Πράξεις που εκτελούνται. Κυρίως από λογική της Control Unit

Ανάγνωση Εντολής (Instruction Fetch)



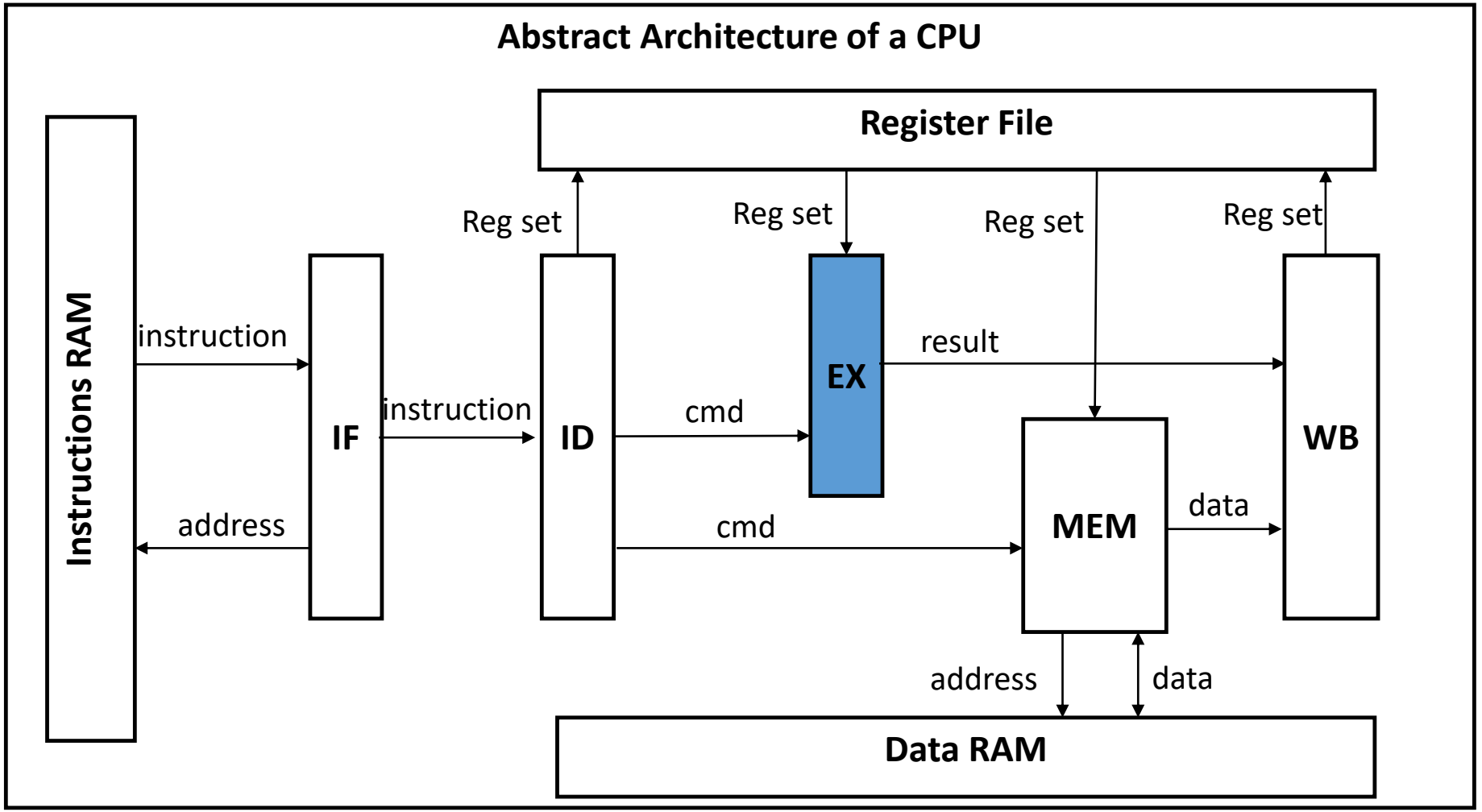
Instruction Fetch (IF): Γίνεται ανάγνωση της επόμενης εντολής (instruction) από την μνήμη εντολών
Αυτό επιτυγχάνεται θέτοντας το **address** στην τιμή του program counter (PC). Θυμηθείτε ο PC κρατάει σε ποια θέση στη μνήμη βρίσκεται η εντολή που εκτελείται

Αποκωδικοποίηση Εντολής (Instruction Decode)



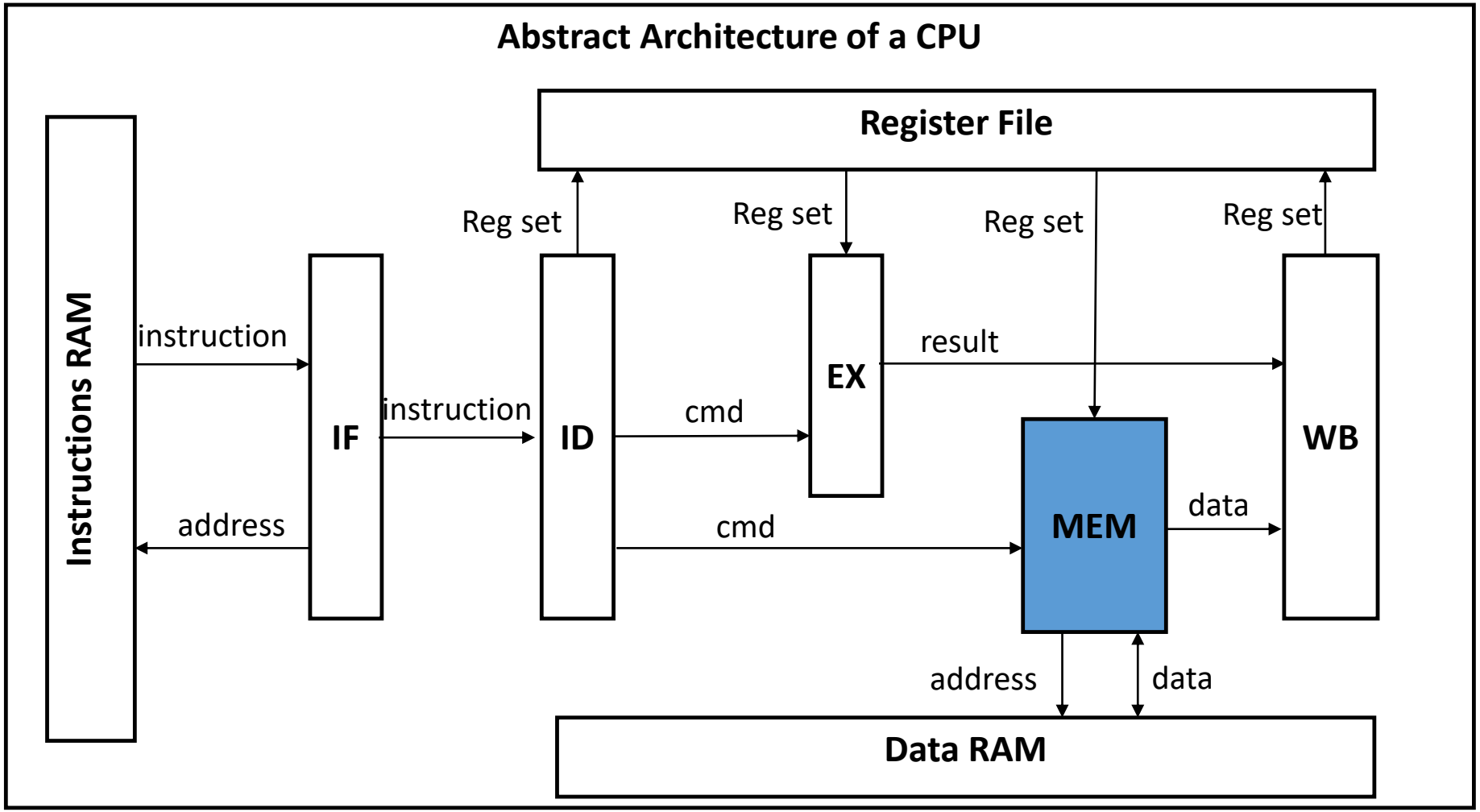
Instruction Decode (ID): ένα κύκλωμα (της μορφής case(instruction)…endcase) ελέγχει για ποια εντολή πρόκειται και ενημερώνει τα επόμενα components για το ποια δεδομένα θα χρειαστούν από την εντολή. Π.χ. αν είναι εντολή που εμπλέκει την μνήμη, τους καταχωρητές ή κάποιο reference/pointer.

Εκτέλεση Εντολής (Instruction Execute)



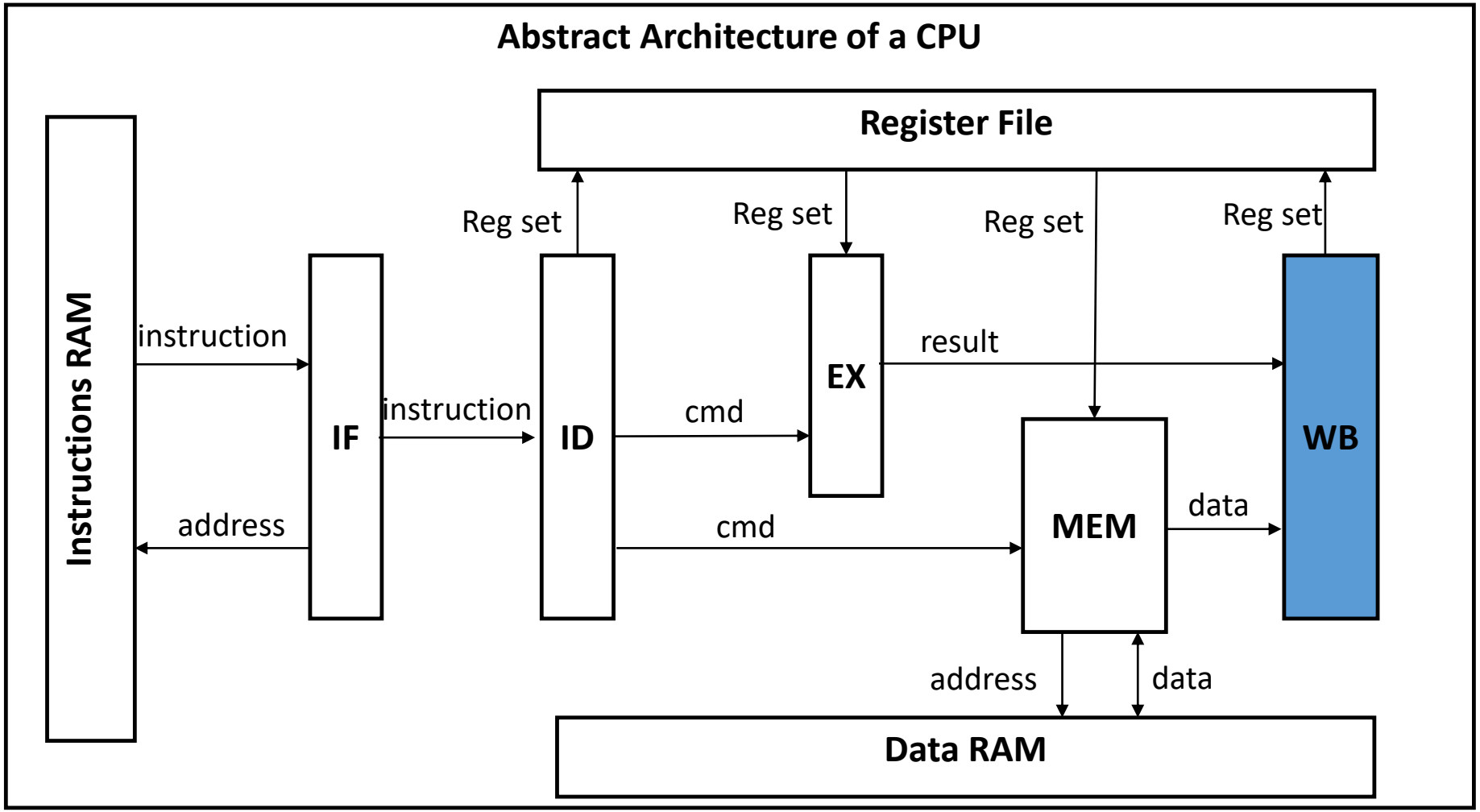
Execute (EX): Εκτελεί εντολές που εμπλέκουν επεξεργασία, όπως αριθμητικές/λογικές πράξεις. Μπορούμε να πούμε ότι οδηγεί την Arithmetic Logic Unit (ALU), η οποία δεν φαίνεται στο σχήμα.

Memory Access/Transactions



Memory Access/Transactions (MEM): ένα κύκλωμα που αναλαμβάνει να διασυνδέσει Register file προς την μνήμη. Από το decode θα ενημερωθεί αν πρόκειται κάποια μεταφορά δεδομένων από καταχωρητές προς την μνήμη. Προσέξτε ότι δεν γράφει αυτό το task καταχωρητές, αλλά στέλνει δεδομένα προς το WB το οποίο αναλαμβάνει να γράψει καταχωρητές.

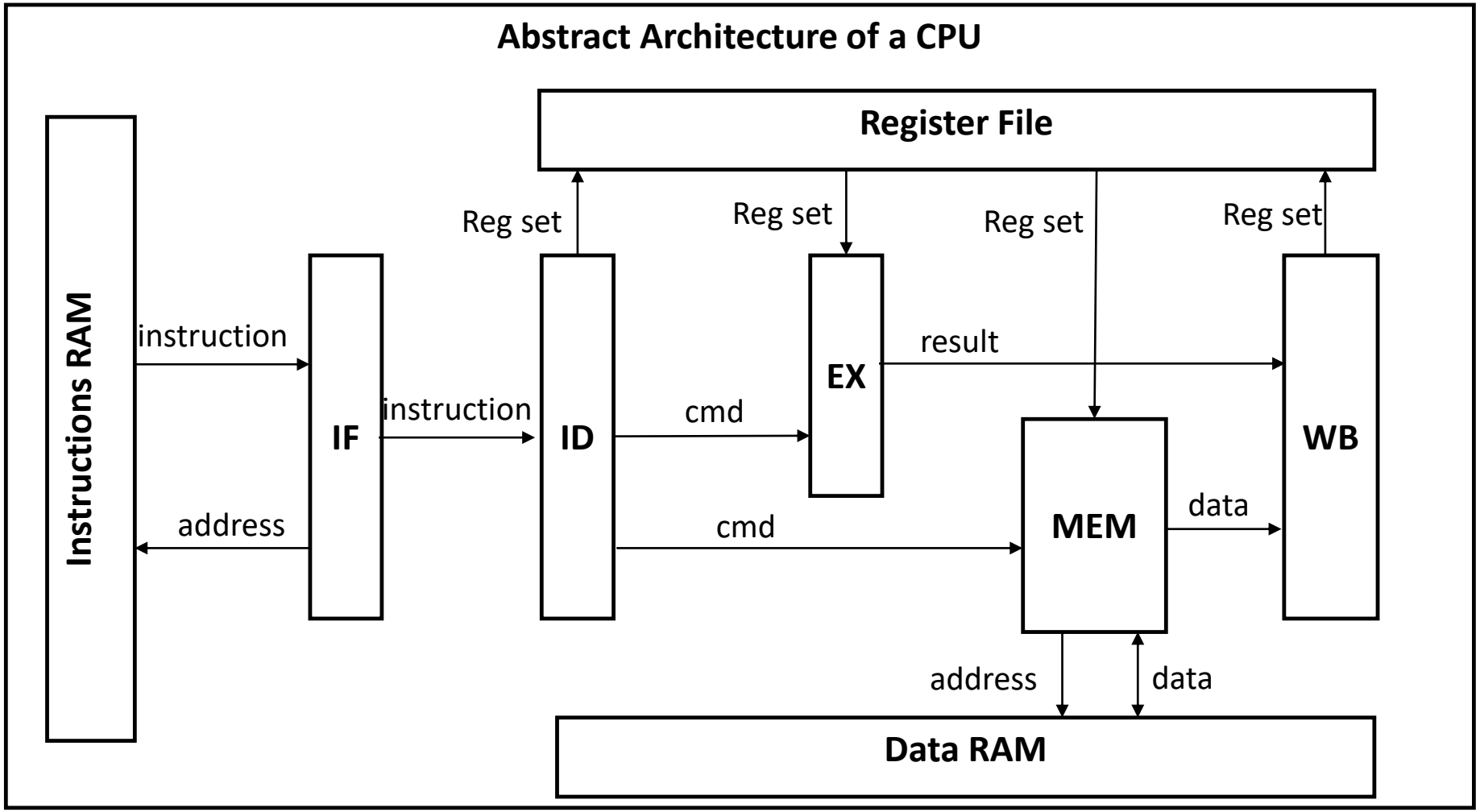
Register Writeback



Register Writeback (WB): αναλαμβάνει να γράψει τους καταχωρητές με αποτελέσματα από το στάδιο EX ή με δεδομένα από την μνήμη.

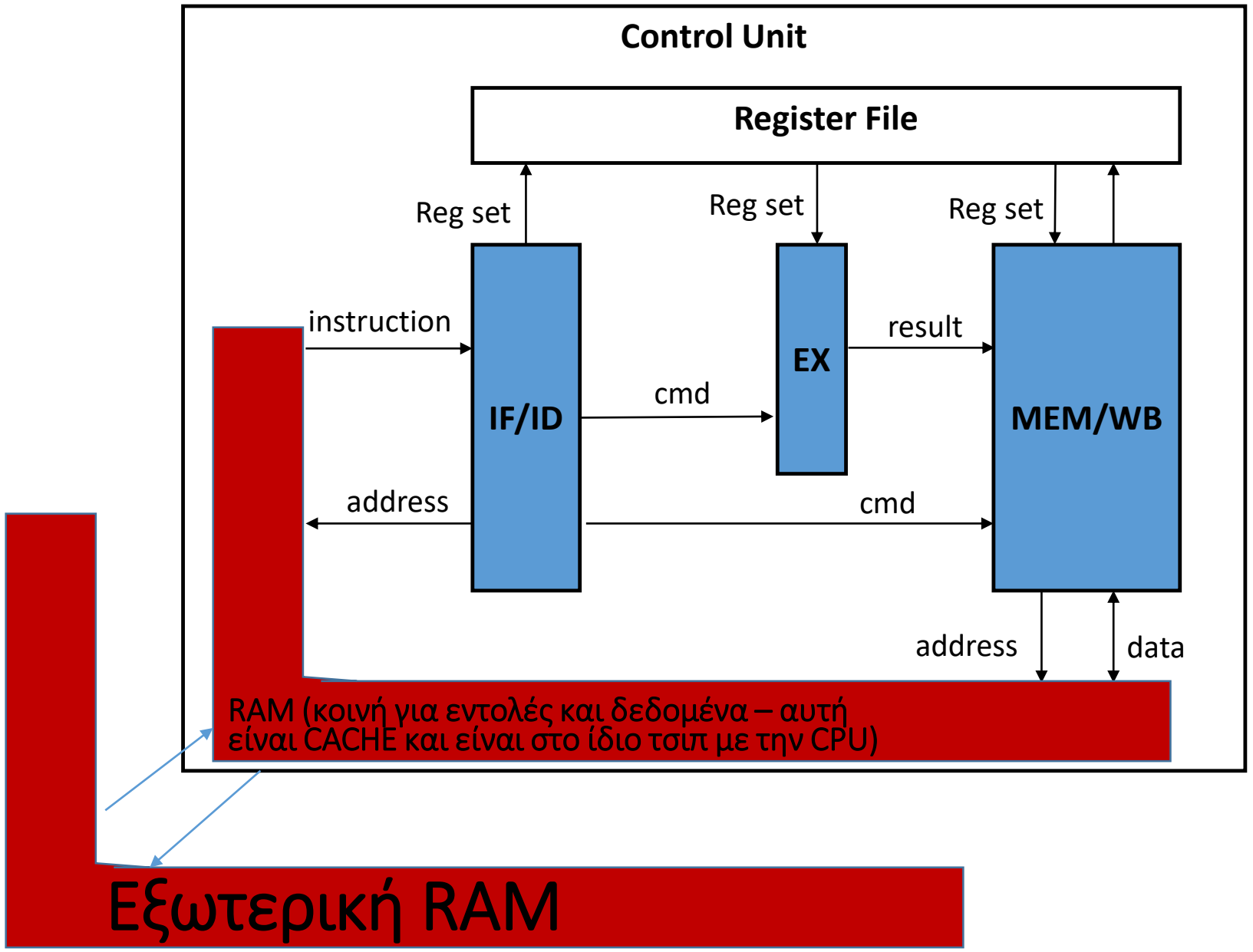
Προσέξτε στο στάδιο αυτό ότι κάποιο αποτέλεσμα της ALU δεν γράφεται στην μνήμη ποτέ αλλά γράφεται στους καταχωρητές και μετά με την επόμενη εντολή προς την μνήμη.

Instruction Fetch

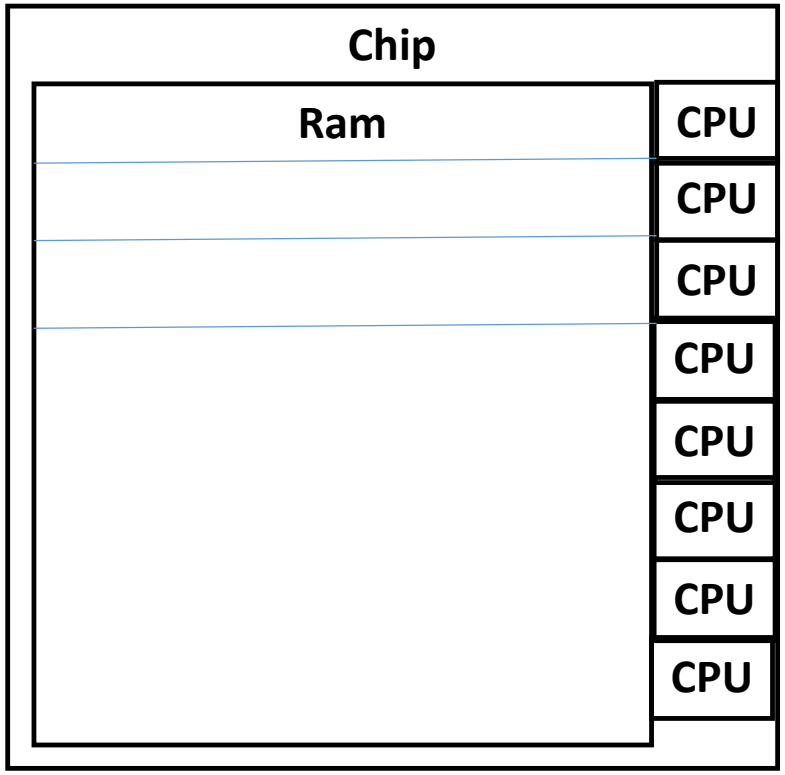


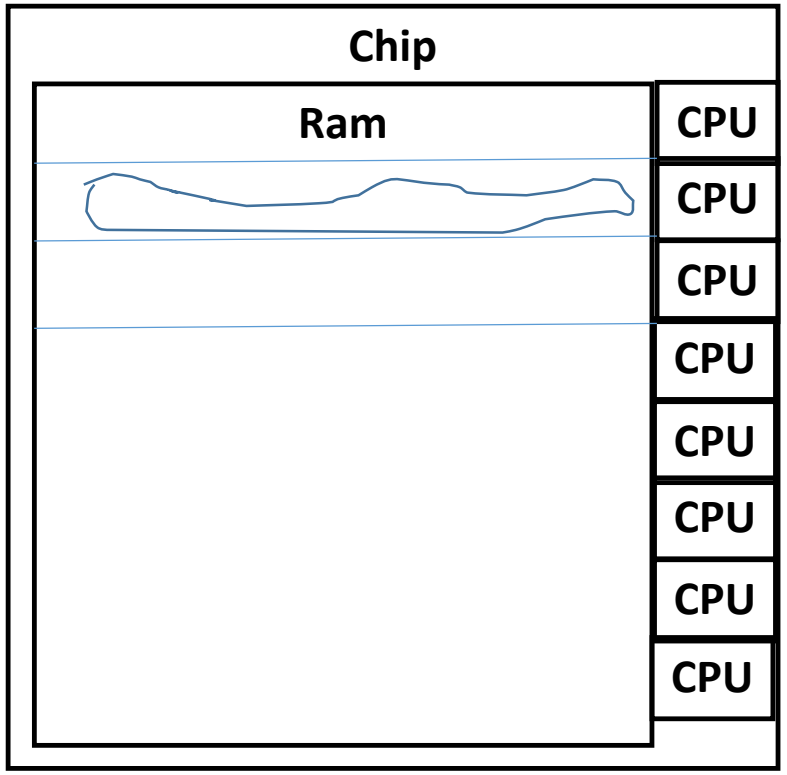
Instruction Decode (ID): ένα κύκλωμα (της μορφής case(instruction)…endcase) ελέγχει για ποια εντολή πρόκειται και ενημερώνει τα επόμενα components για το ποια δεδομένα θα χρειαστούν από την εντολή. Π.χ. αν είναι εντολή που εμπλέκει την μνήμη, τους καταχωρητές ή κάποιο reference/pointer.

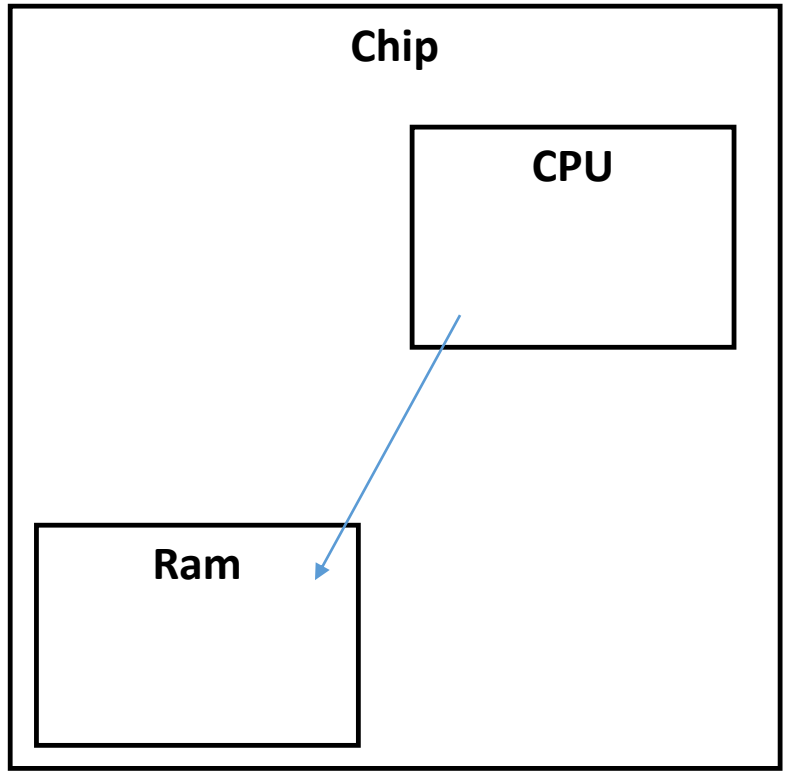
Διοχέτευση τριών σταδίων με κοινή μνήμη για εντολές/δεδομένα



GPUs και τάση προς in-memory computing







Τάση προς in-memory computing – 6T memory cells

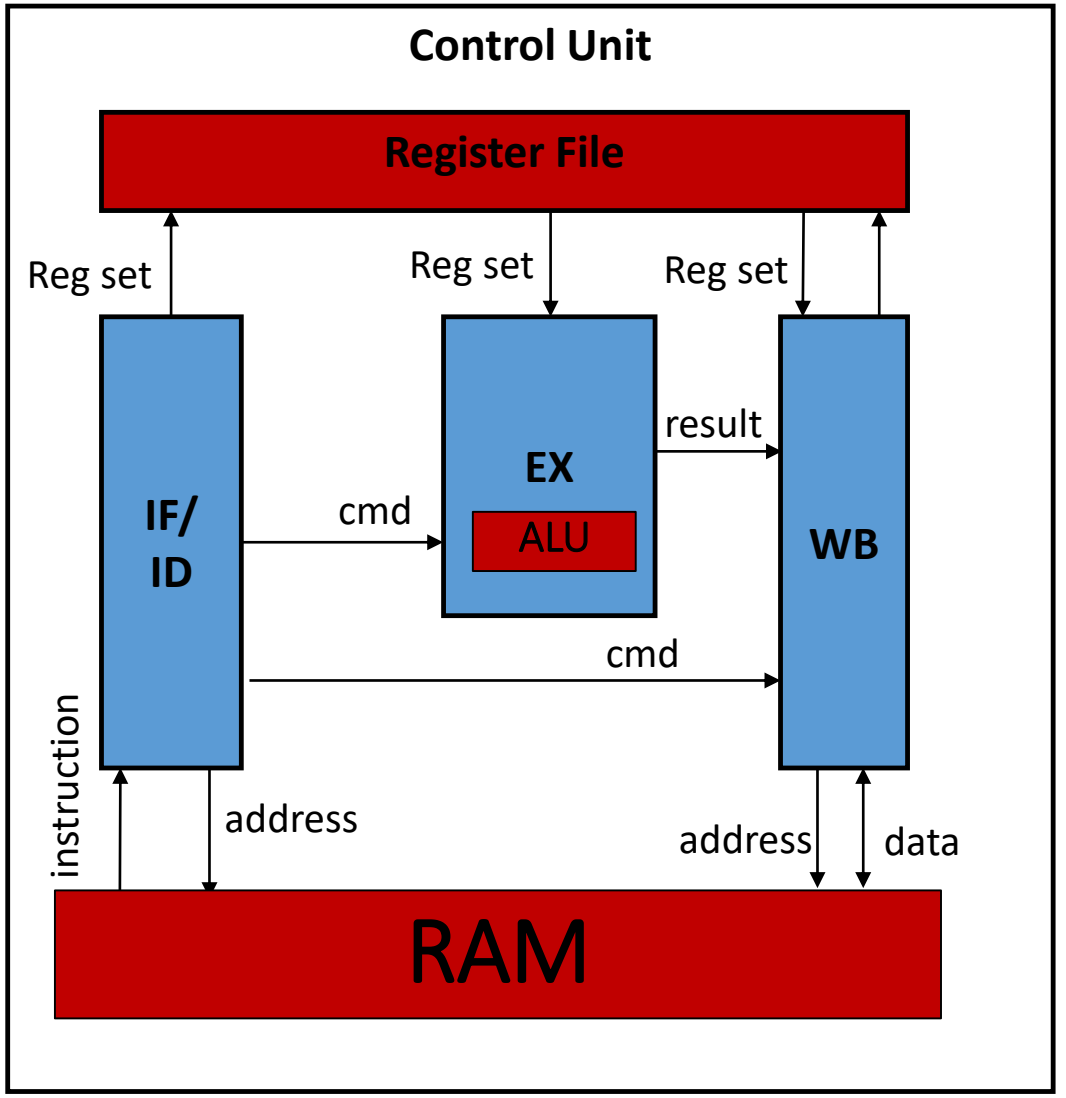
Τάση προς in-memory computing – 6T memory cells

Αρχιτεκτονική του MicroCPU

Ο επεξεργαστής που θα σχεδιάσουμε θα έχει την αρχιτεκτονική διοχέτευσης που φαίνεται εδώ.

Components του επεξεργαστή

Πράξεις που εκτελούνται. Κυρίως από λογική της Control Unit



Μοντελοποίηση της Μνήμης και Σχεδίαση του Ελεγκτή Μνήμης

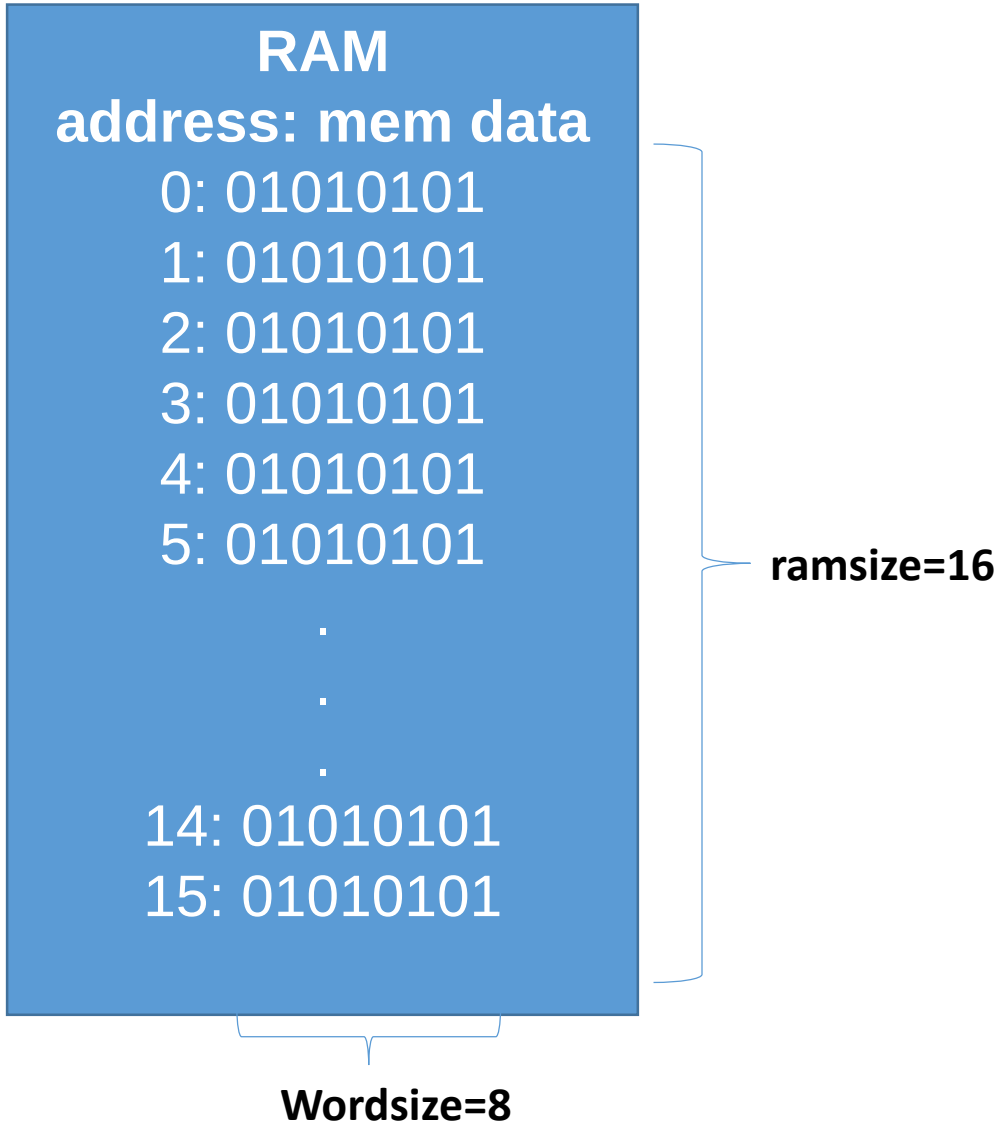
Μοντελοποίηση Μνήμης στην Verilog

Η μνήμη στην Verilog μοντελοποιείται ως καταχωρητές:
reg [wordsize-1:0] my_memory [0:ramsize-1]

Εδώ **wordsize**, που είναι και το **πλάτος της μνήμης**, σημαίνει ότι **μια λέξη** αυτής της μνήμης έχει μέγεθος wordsize bits.

Το **ramsize** είναι το μέγεθος της μνήμης σε πλήθος λέξεων.

Παράδειγμα:
reg [7:0] my_memory [0:15]



Αποθήκευση δεδομένων
my_memory [address]=data_in;

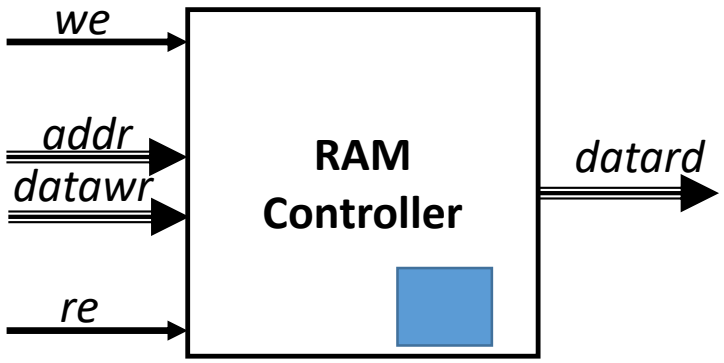
Ανάγνωση δεδομένων
data_in=my_memory[address];

Παρατήρηση:
Συνήθως γίνεται ανάγνωση και αποθήκευση μιας διεύθυνσης τη φορά, γιατί αλλιώς αυξάνει το κόστος της μνήμης.

Ελεγκτής Μνήμης

Ο ελεγκτής ελέγχει την εγγραφή και ανάγνωση της μνήμης.

Παράδειγμα ελεγκτή μνήμης μιας εισόδου εγγραφής και μιας ανάγνωσης:



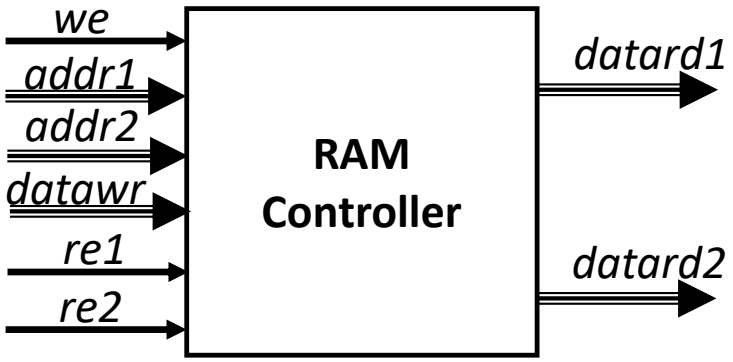

```
module ramcontroller(we, addr, datawr, re, datard);
parameter WORD_SIZE=8;
parameter ADDR_WIDTH=8; //πλήθος bits για διευθύνσεις
//αυτό είναι το μέγεθος της μνήμης είναι δύναμη
//του 2 εις το πλήθος των bits διεύθυνσης
parameter RAM_SIZE=1<<ADDR_WIDTH;
//σήματα write και read enable
input we, re;
//διεύθυνση εγγραφής και ανάγνωσης
input [ADDR_WIDTH-1:0] addr;
input [WORD_SIZE-1:0] datawr;
//σήμα εξόδου για να σταλούν τα δεδομένου που αναγν.
output [WORD_SIZE-1:0] datard;
//η δήλωση της μνήμης
reg [WORD_SIZE-1:0] mem[RAM_SIZE-1:0];

//καταχωρητής για τα δεδομένου που θα διαβαστούν
reg [WORD_SIZE-1:0] datard;

always @ (addr or we or re or datawr)
begin
 if(we)begin
  #2 mem[addr]=datawr;
 end
 if(re) begin
  #2 datard=mem[addr];
 end
end
endmodule
```

Ελεγκτής μνήμης πολλαπλών εισόδων/εξόδων

Μπορούμε να βάλουμε την δυνατότητα πολλαπλών εισόδων/εξόδων για να μπορεί να γίνεται ανάγνωση και εγγραφή από και προς πολλές πηγές ταυτόχρονα.

Αυτό όμως αυξάνει την πολυπλοκότητα και το κόστος μιας μνήμης

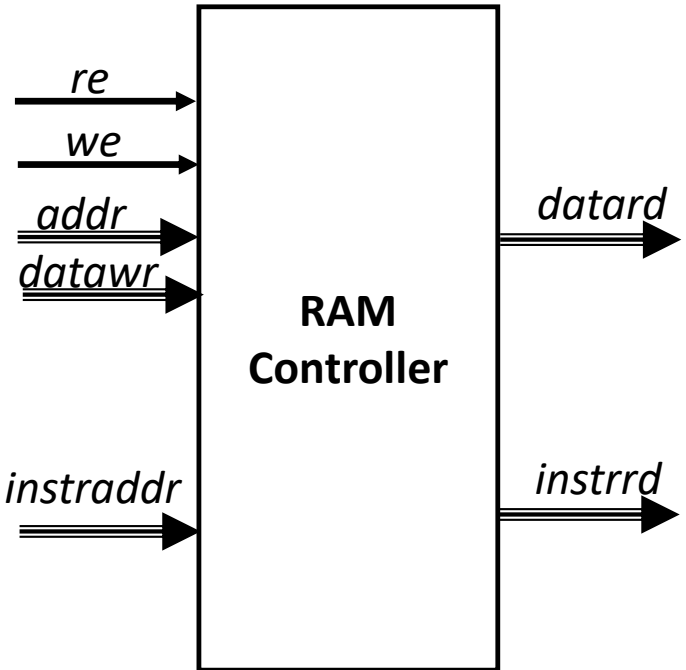
Παράδειγμα ελεγκτή μνήμης μιας εισόδου εγγραφής και δύο εισόδων ανάγνωσης:



Ελεγκτής μνήμης πολλαπλών εισόδων/εξόδων

Θα χρησιμοποιήσουμε μια μνήμη με έναν ελεγκτή μιας εισόδου εγγραφής και δύο εισόδων ανάγνωσης για τον επεξεργαστή μας. Την δυνατότητα εισόδου θα την χρησιμοποιήσουμε για να γράφουμε δεδομένα στην μνήμη και τις δύο θύρες ανάγνωσης θα τις χρησιμοποιούμε, την μία για να διαβάζουμε εντολές προς εκτέλεση και την άλλη δεδομένα… ΝΑΙ και οι εντολές αποθηκεύονται στην μνήμη.

Παράδειγμα ελεγκτή μνήμης μιας εισόδου εγγραφής και δύο εισόδων ανάγνωσης. Η μία είναι δεδομένων και η άλλη εντολών/instructions:



Μπλοκ για δεδομένα

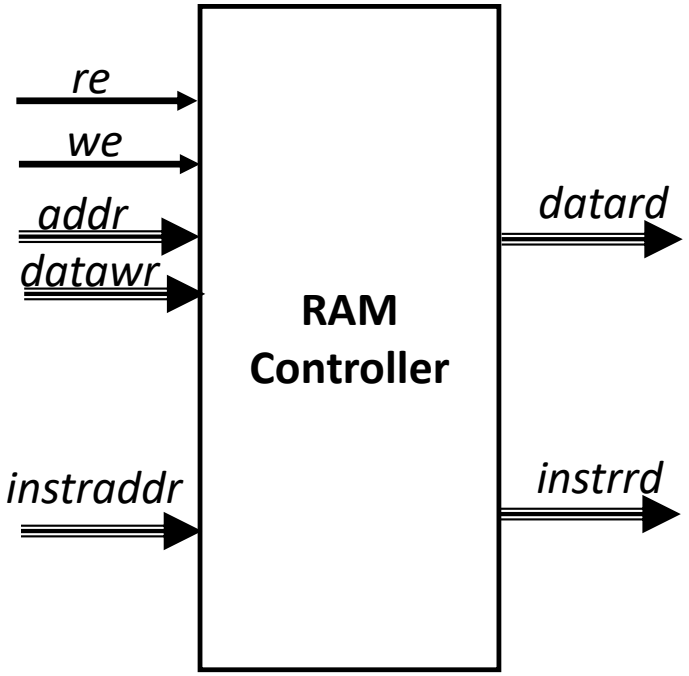
```
always @ (addr or we or re or datawr)
begin
  if(we)begin
    mem[addr]=datawr;
  end
  if(re) begin
    datard=mem[addr];
  end
end
```

Μπλοκ για instructions

```
always @ (instraddr)
begin
   instrrd=mem[instraddr];
end
```

Η Μνήμη του επεξεργαστή MicroCPU

```
module MCPU_RAMController(we, datawr, re, addr, datard, instraddr, instrrd);
parameter WORD_SIZE=8;
parameter ADDR_WIDTH=8;
parameter RAM_SIZE=1<<ADDR_WIDTH;
input we, re;
input [WORD_SIZE-1:0] datawr;
input [ADDR_WIDTH-1:0] addr;
input [ADDR_WIDTH-1:0] instraddr;
output [WORD_SIZE-1:0] datard;
output [WORD_SIZE-1:0] instrrd;

reg [WORD_SIZE-1:0] mem[RAM_SIZE-1:0];
reg [WORD_SIZE-1:0] datard;
reg [WORD_SIZE-1:0] instrrd;

always @ (addr or we or re or datawr)
begin
  if(we)begin
    mem[addr]=datawr;
  end
  if(re) begin
    datard=mem[addr];
  end
end
always @ (instraddr)
begin
    instrrd=mem[instraddr];
end

endmodule
```



Άσκηση 6.1a: Να γράψετε ένα testbench με το οποίο να δίνετε τυχαία δεδομένα για εγγραφή και ανάγνωση στον ελεγκτή μνήμης του MicroCPU και προσομοιώνοντάς το στο Modelsim να βεβαιώσετε την ορθότητα του σχεδιασμού. Προσοχή μπορεί να υπάρχει λάθος στον κώδικα.

Αρχικοποίηση μνήμης από αρχείο (κατά την προσομοίωση)

Κατά την προσομοίωση η μνήμη μπορεί να αρχικοποιηθεί από αρχείο με τα tasks:
$readmemb και $readmemh.

Το **$readmemb** διαβάζει αρχείο που περιγράφει τα δεδομένα τις μνήμης σε δυαδική αναπαράσταση και το **$readmemh** διαβάζει αρχείο που περιγράφει τα δεδομένα τις μνήμης σε δεκαεξαδική αναπαράσταση.

Σύνταξη:
$readmemh("file_name",mem_array, start_addr, stop_addr);
Σημείωση: τα start_addr και stop_addr είναι προαιρετικά.

Παράδειγμα χρήσης task $readmemb:

```
module  memory();
 reg [7:0] my_memory [0:255];
 initial begin
  $readmemb("memory.list", my_memory);
 end
endmodule
```

Παράδειγμα αρχείου "memory.list":

```
//Comments are allowed
1100_1100   // This is first address i.e 8'h00
1010_1010   // This is second address i.e 8'h01
@ 55        // Jump to new address 8'h55
0101_1010   // This is address 8'h55
0110_1001   // This is address 8'h56
```

Αρχικοποίηση μνήμης από αρχείο (κατά την προσομοίωση)

```
module registerfile(ifconf, wbbit, wbdirection, alu1, alu2, fromwbdata, towbdata);
parameter BITS_NUMBER=8;
//configuration from FETCH
input [1:0] ifconf;
//γράψτε το process που θα υλοποιεί αυτής της ανάθεσης καταχωρητών
//4 register and one PC
output [BITS_NUMBER-1:0] alu1; // to be the first operand of alu
output [BITS_NUMBER-1:0] alu2; // to be the second operand of alu
output [BITS_NUMBER-1:0] towbdata; //this is data to be written to memory
//this processor has 4 registers
reg [BITS_NUMBER-1:0] R[3:0];
wire [BITS_NUMBER-1:0] r0,r1,r2,r3;
assign r0=REGS[0];
assign r1=REGS[1];
assign r2=REGS[2];
assign r3=REGS[3];
```

```
//if CONF_R0_R1:ALU1 R0 and ALU2 is R1
//if CONF_R0_R2:ALU1 R0 and ALU2 is R2
//if CONF_R0_R3:ALU1 R0 and ALU2 is R3
//if CONF_R1_R2:ALU1 R1 and ALU2 is R2
//if CONF_R1_R3:ALU1 R1 and ALU2 is R3
//if CONF_R2_R3:ALU1 R2 and ALU2 is R3
//if wbbit:
  //if wbdirection==0
    //write data from fromwbdata to R0
  //else if wbdirection==1
     //write data from R0 to towbdata
parameter  [NEEDED_BITS:0] CONF_R0_R1  = 0;
parameter  [NEEDED_BITS:0] CONF_R0_R2  = 1;
parameter  [NEEDED_BITS:0] CONF_R0_R3  = 2;
parameter  [NEEDED_BITS:0] CONF_R1_R2  = 3;
parameter  [NEEDED_BITS:0] CONF_R1_R3 = 4;
parameter  [NEEDED_BITS:0] CONF_R2_R3  = 5;
always @ (ifconf, wbbit, wbdirection)
begin
  if(...)
end
endmodule
```

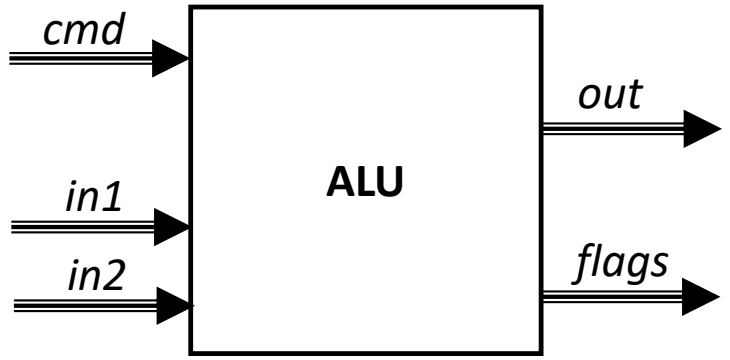
Σχεδίαση της Αριθμητικής Λογικής Μονάδας – Arithmetic Logic Unit (ALU)

Βασικά για την ALU

Η αριθμητική/λογική μονάδα είναι υπεύθυνη για την επεξεργασία των δεδομένων στον επεξεργαστή.

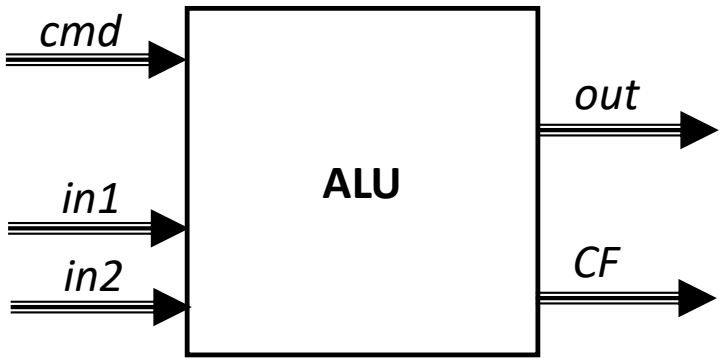
Εκτελεί αριθμητικές (πρόσθεση, αφαίρεση, πολλαπλασιασμό κτλ) και λογικές πράξεις (bitwise not, or, xor κτλ.) με τους καταχωρητές.

Επιστρέφει τα δεδομένα της σε κάποιον καταχωρητή και τα flags από τις αριθμητικές πράξεις (π.χ. κρατούμενα υπολογισμών carry flag, διαίρεσης με το μηδέν (zero flag) κτλ.)



Η ALU του MicroCPU

```
module MCPU_Alu(cmd,in1,in2,out,CF);
parameter CMD_SIZE=4;
parameter WORD_SIZE=8;

parameter  [1:0]  CMD_AND   = 0; //2'b00
parameter  [1:0]  CMD_OR    = 1; //2'b01
parameter  [1:0]  CMD_XOR   = 2; //2'b10
parameter  [1:0]  CMD_ADD   = 3; //2'b11

input [CMD_SIZE-1:0] cmd;
input [WORD_SIZE-1:0] in1;
input [WORD_SIZE-1:0] in2;
output[WORD_SIZE-1:0] out;
//carry flag
output CF;
wire [CMD_SIZE-1:0] cmd;
wire [WORD_SIZE-1:0] in1;
wire [WORD_SIZE-1:0] in2;
reg  [WORD_SIZE-1:0] out;
//carry flag
reg  CF;
always @ (cmd, in1, in2)
  #2
case(cmd)
  CMD_AND : begin
             out = in1&in2;
     end
  CMD_OR : begin
            out = in1|in2;
           end
  CMD_XOR : begin
             out = in1^in2;
           end
  default : begin
            {CF,out} = in1+in2;
           end
 endcase
endmodule
```



Άσκηση 6.1b: Να γράψετε ένα testbench με το οποίο να δίνετε τυχαίες εντολές και operands στην ALU του MicroCPU και προσομοιώνοντάς το στο Modelsim να βεβαιώσετε την ορθότητα του σχεδιασμού. Προσοχή μπορεί να υπάρχει λάθος στον κώδικα.

Το σύνολο των εντολών – Instructions Set

Εντολές μηχανής (instruction set) του MicroCPU

- **Κλασσικές εντολές επεξεργασίας** που εμπλέκουν την ALU για αριθμητικές και λογικές πράξεις, π.χ.:
  AND operand1 operand2 operand3
  OR operand1 operand2 operand3
  XOR operand1 operand2 operand3
  ADD operand1 operand2 operand3
  Το αποτέλεσμα γράφεται στον operand1. Οι πράξεις εμπλέκουν δύο operands τους operand2 και operand3.
  Operands για τον MicroCPU είναι μόνο οι καταχωρητές του.
- **Εντολές μετακίνησης δεδομένων από καταχωρητή σε καταχωρητή**:
  MOV Rd R
  τα δεδομένα αντιγράφονται από τον καταχωρητή R στον καταχωρητή Rd (το Rd προκύπτει από το register destination).
- **Εντολές φόρτωσης δεδομένων από μνήμη σε καταχωρητή**:
  Load_FROM_MEM Rd address
  αντιγράφει τα δεδομένα από την διεύθυνση μνήμης address στον καταχωρητή Rd.
- **Εντολές αποθήκευσης δεδομένων από καταχωρητή στην μνήμη**:
  STORE_TO_MEM address R
  αποθηκεύει στη διεύθυνση μνήμης address τα δεδομένα του καταχωρητή R.
- **Εντολές Αρχικοποίησης τιμών**:
  OP_SHORT_TO_REG Rd value
  αντιγράφει την τιμή value των 8 bits στον καταχωρητή Rd.
- **Εντολές διακλάδωσης:**
  BNZ Rc address
  μετακινεί τον program counter στην τιμή address αν ο καταχωρητής Rc δεν είναι 0. (Branch if Not Zero)

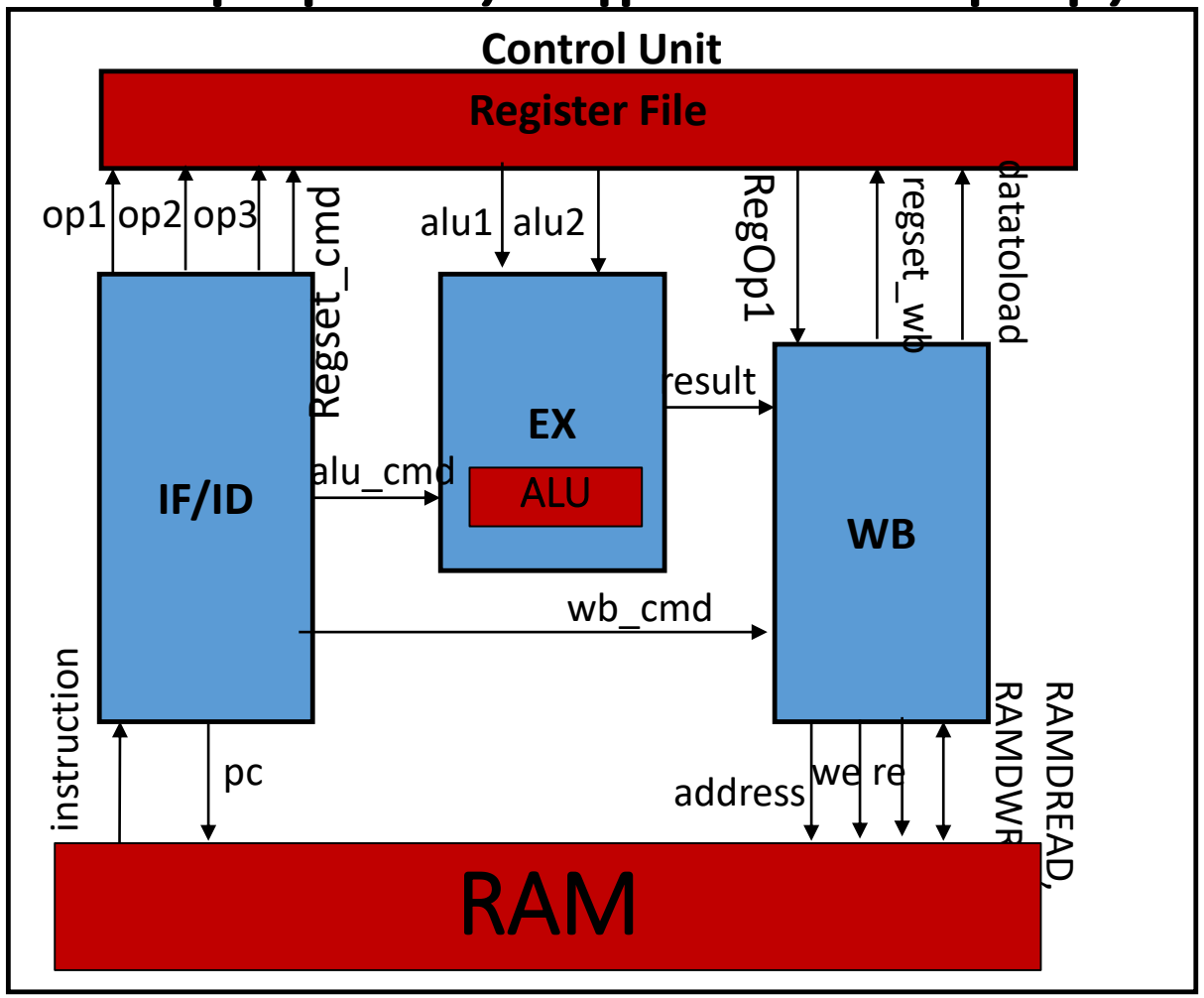
Σχεδίαση του αρχείου καταχωρητών – Register File

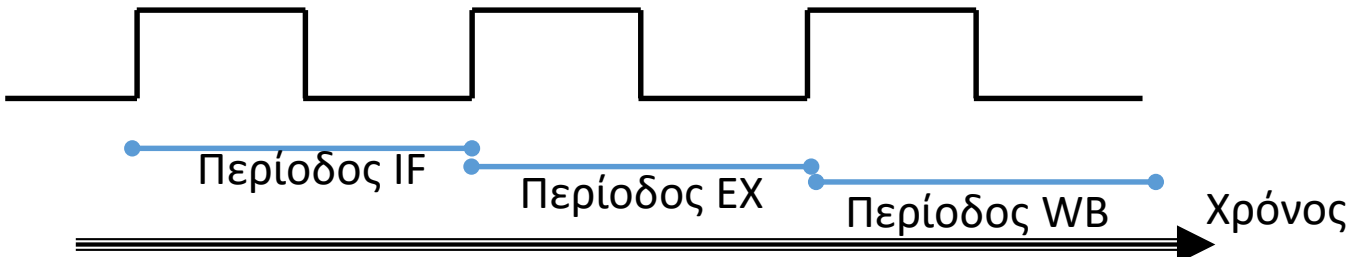
Το αρχείο καταχωρητών – register file

Το αρχείο καταχωρητών του MicroCPU έχει μόνο 4 καταχωρητές R0,R1,R2 και R3 των 16 bits.

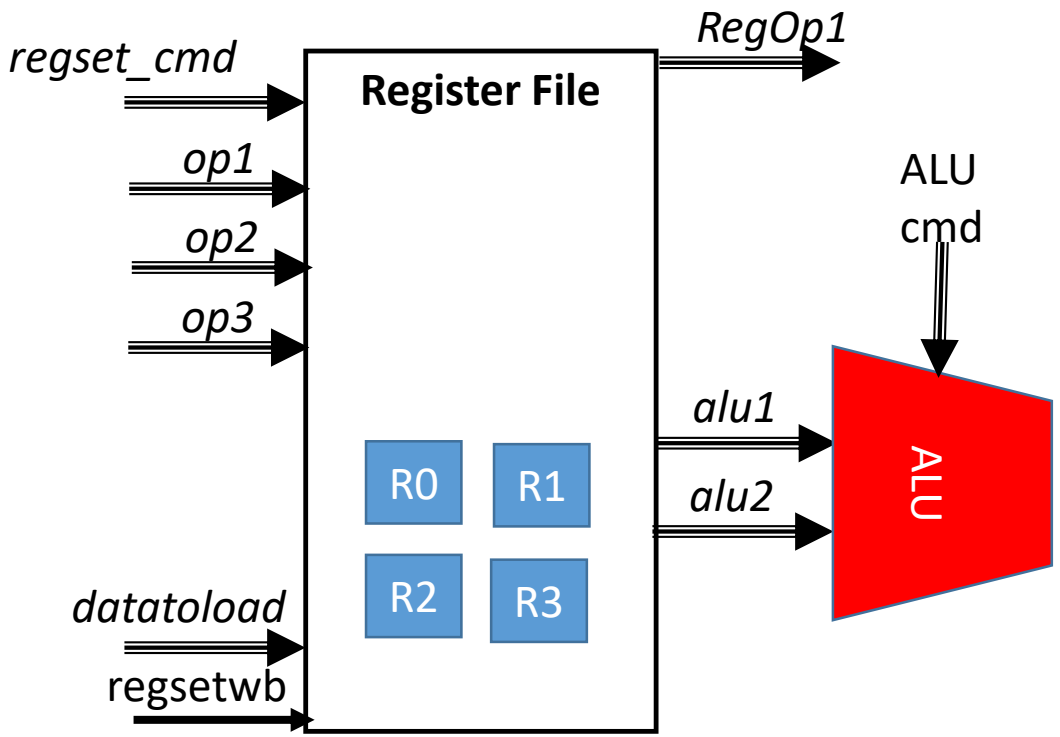
Τουλάχιστον ένας καταχωρητής συμμετέχει σε κάθε εντολή
AND Rd Ra Rb
OR Rd Ra Rb
XOR Rd Ra Rb
ADD Rd Ra Rb
MOV Rd R
Load_FROM_MEM Rd address
STORE_TO_MEM address R
OP_SHORT_TO_REG Rd value
BNZ Rc address

Λεπτομέρειες σηματοδότησης του register file





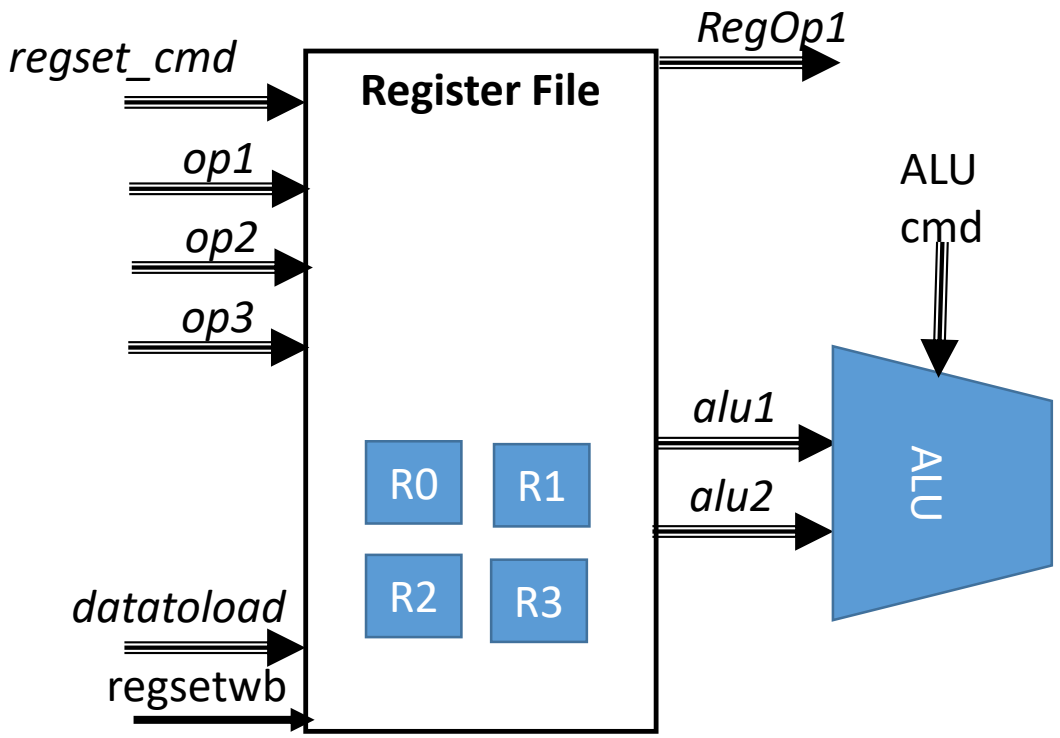
ADD operand1 operand2 operand3



Οι λειτουργίες του αρχείου καταχωρητών

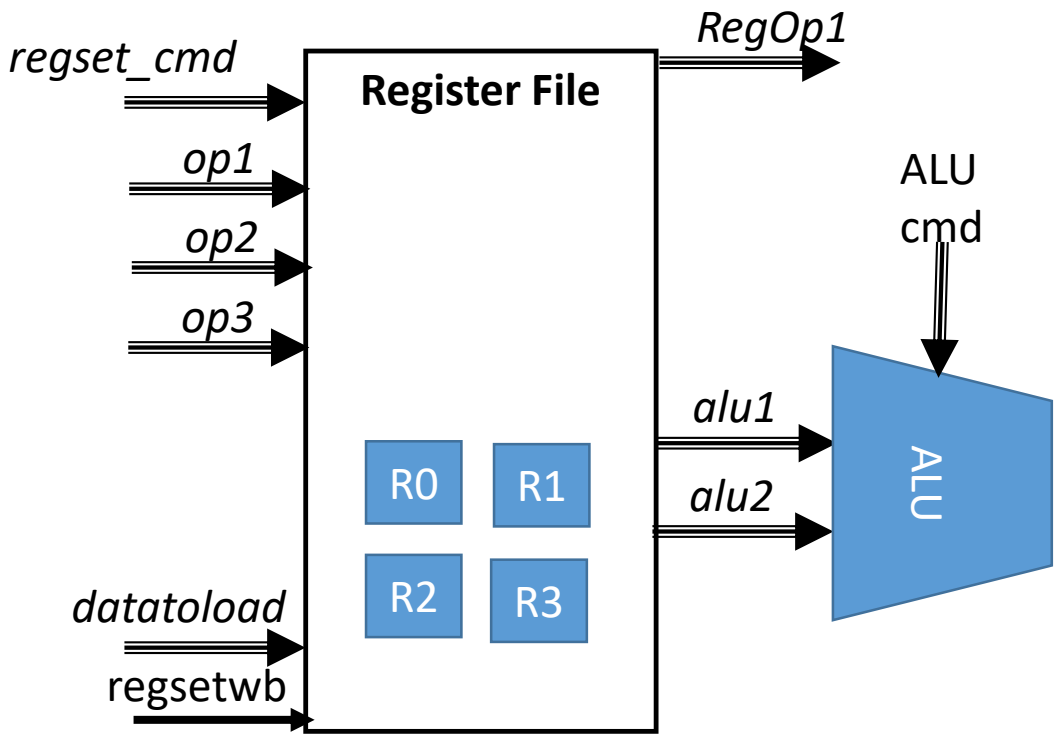
Πάντα εκτελεί αυτόν τον κώδικα:
assign RegOp1=R[op1[REGS_NUMBER_WIDTH-1:0]];
assign alu1=R[op2[REGS_NUMBER_WIDTH-1:0]];
assign alu2=R[op3[REGS_NUMBER_WIDTH-1:0]];

Και εκτελεί τις παρακάτω λειτουργίες, με ασύγχρονη ενεργοποίηση κατά το wb.
NORMAL_EX:
MOV_INTERNAL:
LOAD_FROM_DATA:
DO_NOTHING:



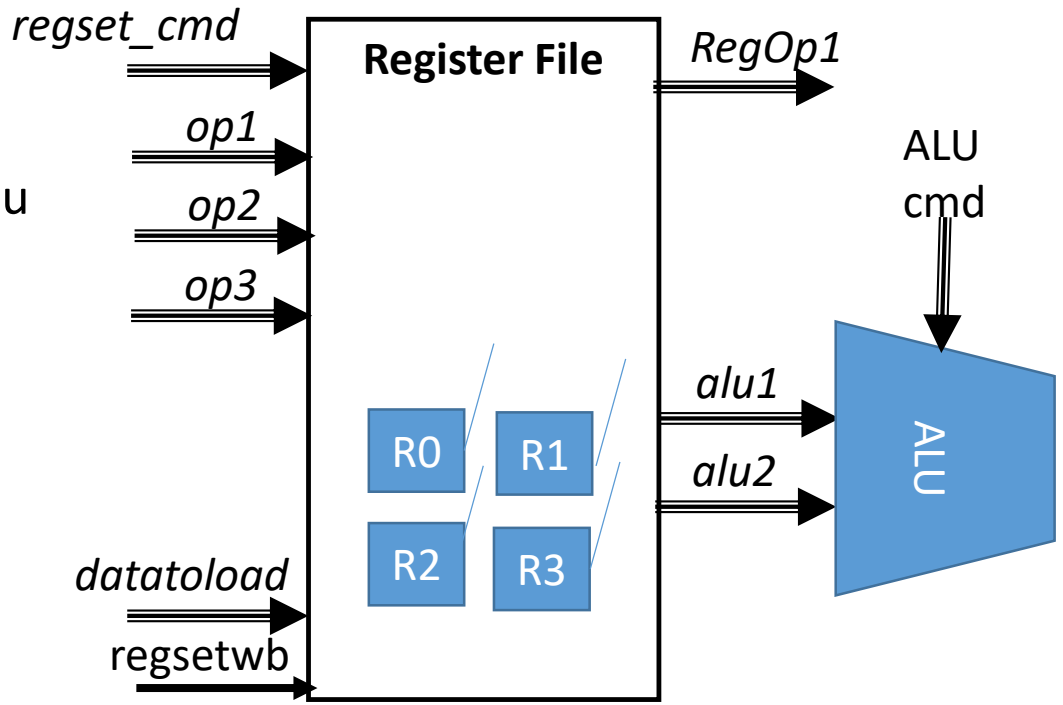
Οι λειτουργίες του αρχείου καταχωρητών

```
module MCPU_Registerfile(op1, op2, op3, RegOp1, alu1, alu2,
datatoload, regsetwb, regsetcmd);
parameter WORD_SIZE=8;
parameter OPERAND_SIZE=4;
parameter REGS_NUMBER_WIDTH=2;
parameter REGISTERS_NUMBER=1<<REGS_NUMBER_WIDTH;

input [OPERAND_SIZE-1:0] op1,op2,op3;
input [WORD_SIZE-1:0] datatoload;

input [1:0] regsetcmd;
input regsetwb;
//REGISTER FILE COMMAND (regsetcmd control bits)
parameter  [1:0]  NORMAL_EX  = 0; //2'b00
parameter  [1:0]  MOV_INTERNAL   = 1; //2'b01
parameter  [1:0]  LOAD_FROM_DATA   = 2; //2'b10
parameter  [1:0]  DO_NOTHING   = 3; //2'b11

//these are always set outputs implemented structurally.
```



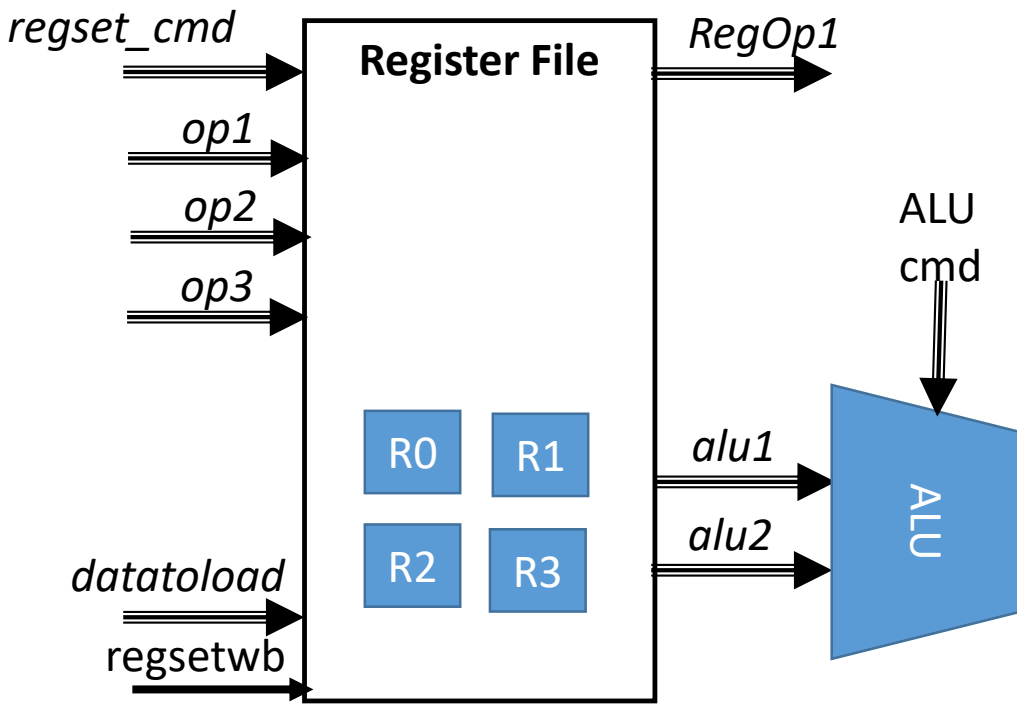
Οι λειτουργίες του αρχείου καταχωρητών

```
//these are always set outputs implemented structurally.
// correspond to translated operands to registers
//operand1 is translated to RegOp1
//operand2 is translated to alu1
//operand3 is translated to alu2
output [WORD_SIZE-1:0] RegOp1;
output [WORD_SIZE-1:0] alu1; // to be the first operand of alu
output [WORD_SIZE-1:0] alu2; // to be the second operand of alu
//this processor has 4 registers
reg [WORD_SIZE-1:0] R[REGISTERS_NUMBER-1:0];
//this processor has 4 registers //aliases
wire [WORD_SIZE-1:0] r0,r1,r2,r3;
//alias
assign r0=R[0];
assign r1=R[1];
assign r2=R[2];
assign r3=R[3];
```



Οι λειτουργίες του αρχείου καταχωρητών

```
//operands translation to registers
assign RegOp1=R[op1[REGS_NUMBER_WIDTH-1:0]];
assign alu1=R[op2[REGS_NUMBER_WIDTH-1:0]];
assign alu2=R[op3[REGS_NUMBER_WIDTH-1:0]];
//whenever this unit needs to act - this signal is asserted at WB
//from the control unit
always @(posedge regsetwb)
begin
 #1
 case(regsetcmd)
 NORMAL_EX,LOAD_FROM_DATA:
 begin
  R[op1[REGS_NUMBER_WIDTH-1:0]]<=datatoload;
 end
 MOV_INTERNAL:
 begin
  R[op1[REGS_NUMBER_WIDTH-1:0]]<=R[op2[REGS_NUMBER_WIDTH-1:0]];
 end
 default:
 begin
 end
 endcase
end
endmodule
```

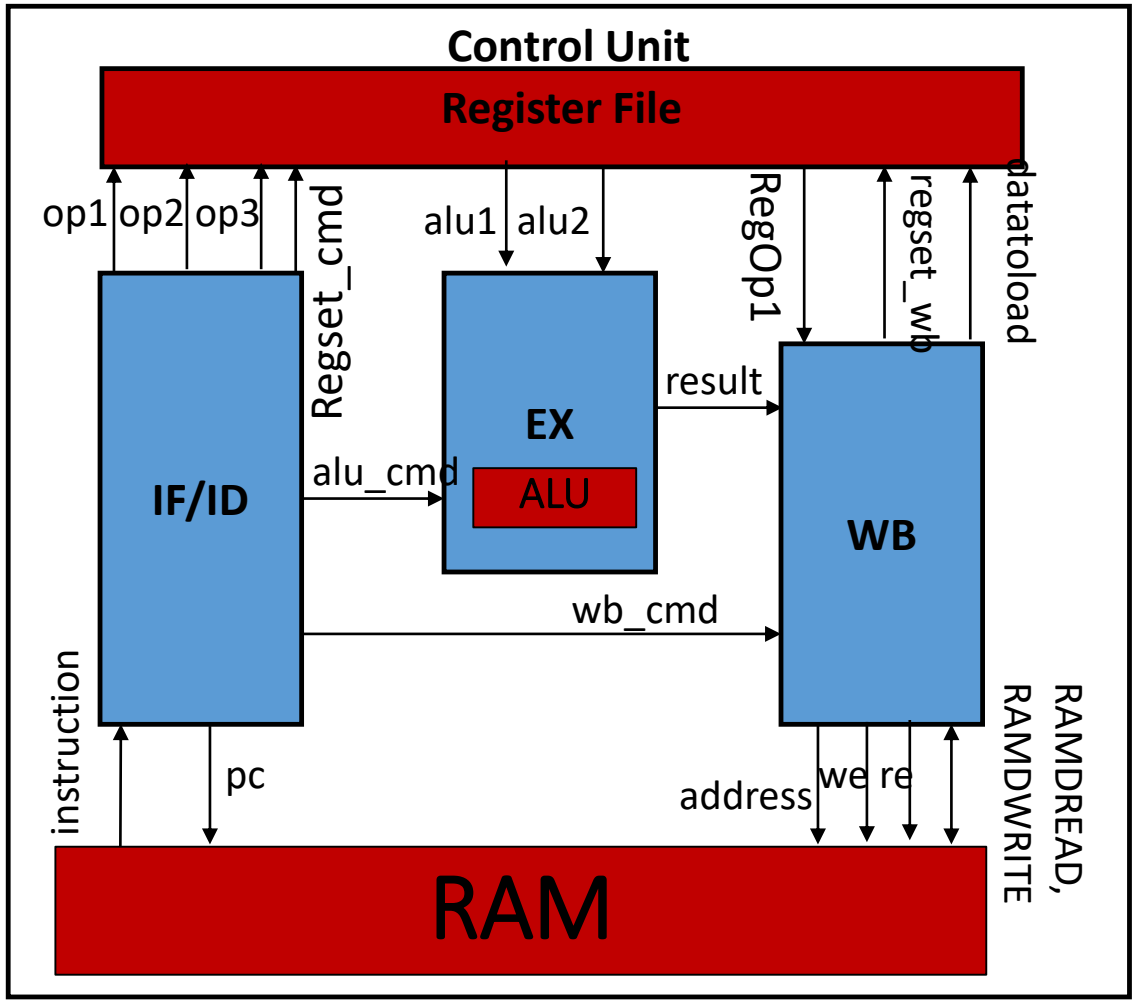


Mov R2 R3

Η μονάδα ελέγχου – Control Unit

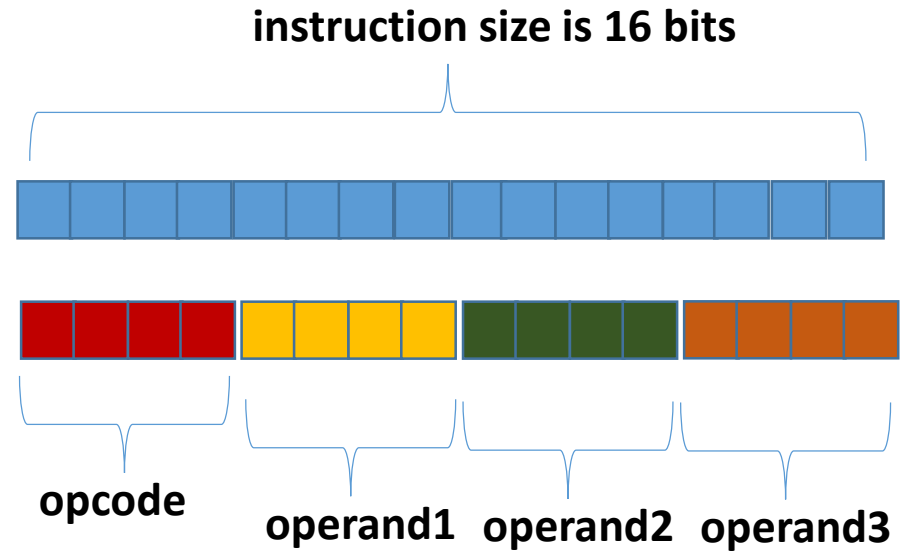
Οι λειτουργίες της μονάδας ελέγχου

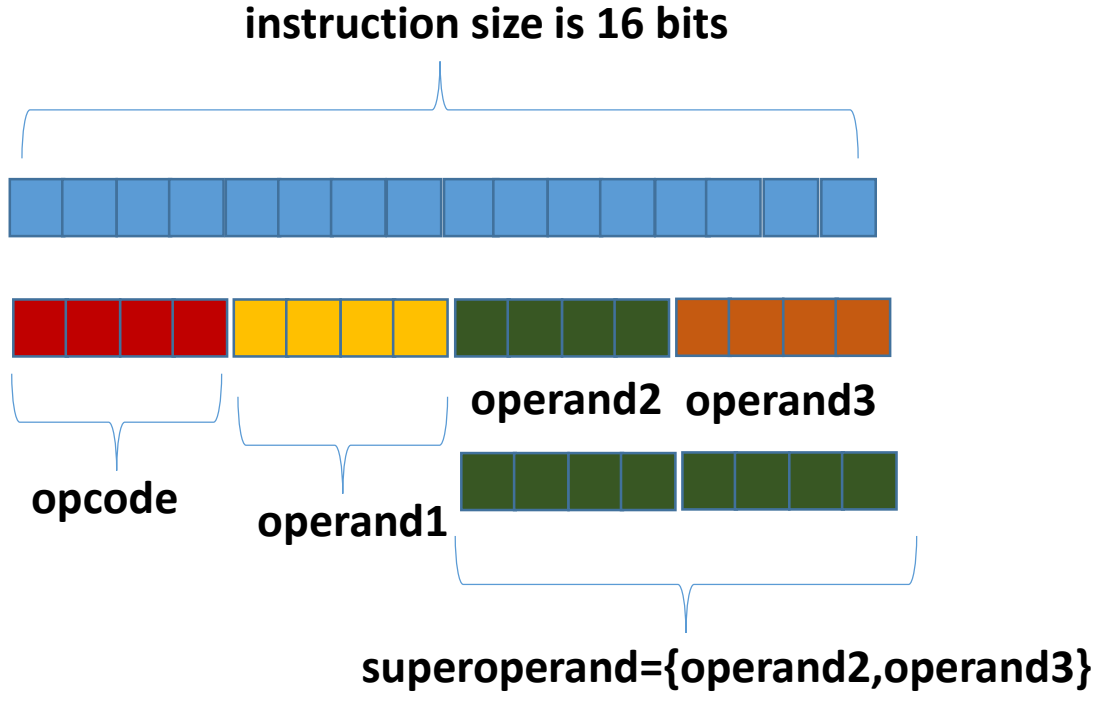
Είναι υπεύθυνη για την αλλαγή των καταστάσεων της διοχέτευσης και την κατάλληλη σηματοδότηση των μονάδων (ALU, registerfile, RAM) την κατάλληλη χρονική στιγμή.



Η μονάδας ελέγχου του MicroCPU – κωδικοποίηση/αποκωδικοποίηση εντολών

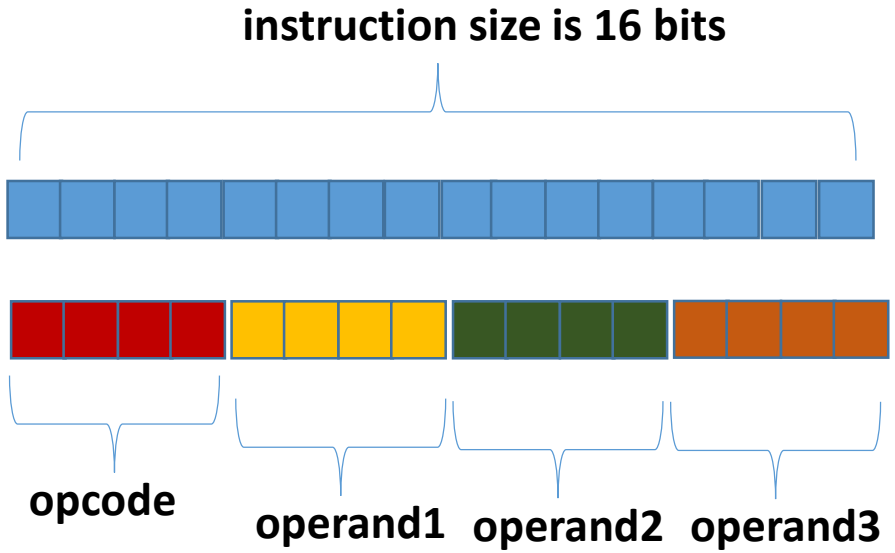
```
module MCPU(clk, reset);
parameter WORD_SIZE=16; //WORD_SIZE is the word size of our
processor. Each word is 16 bits or 2 bytes
parameter ADDR_WIDTH=8; //that means 2^8=256 size of memory of
WORD_SIZE word
parameter OPCODE_SIZE=4;
parameter ALU_CMD_SIZE=2;
parameter OPERAND_SIZE=4;

//instructions will have the following structure:
//OPCODE OPERAND1 OPERAND2 OPERAND3
parameter INSTRUCTION_SIZE = OPCODE_SIZE + OPERAND_SIZE*3;

wire [INSTRUCTION_SIZE-1:0] instruction;

input clk;
input reset;
wire [OPCODE_SIZE-1:0] opcode;
wire [OPERAND_SIZE-1:0] operand1;
wire [OPERAND_SIZE-1:0] operand2;
wire [OPERAND_SIZE-1:0] operand3;
```





Η μονάδας ελέγχου του MicroCPU – κωδικοποίηση εντολών – opcode options – instruction set

```
//opcodes for the supported instruction set
parameter  [OPCODE_SIZE-1:0]  OP_AND  = 0; //4'b0000
parameter  [OPCODE_SIZE-1:0]  OP_OR   = 1; //4'b0001
parameter  [OPCODE_SIZE-1:0]  OP_XOR  = 2; //4'b0010
parameter  [OPCODE_SIZE-1:0]  OP_ADD  = 3; //4'b0011

parameter  [OPCODE_SIZE-1:0]  OP_MOV  = 4; //4'b0100
parameter  [OPCODE_SIZE-1:0]  OP_LOAD_FROM_MEM = 5; //4'b0101
parameter  [OPCODE_SIZE-1:0]  OP_STORE_TO_MEM = 6; //4'b0110
parameter  [OPCODE_SIZE-1:0]  OP_SHORT_TO_REG = 7; //4'b0111
parameter  [OPCODE_SIZE-1:0]  OP_BNZ  = 8; //4'b1000
```

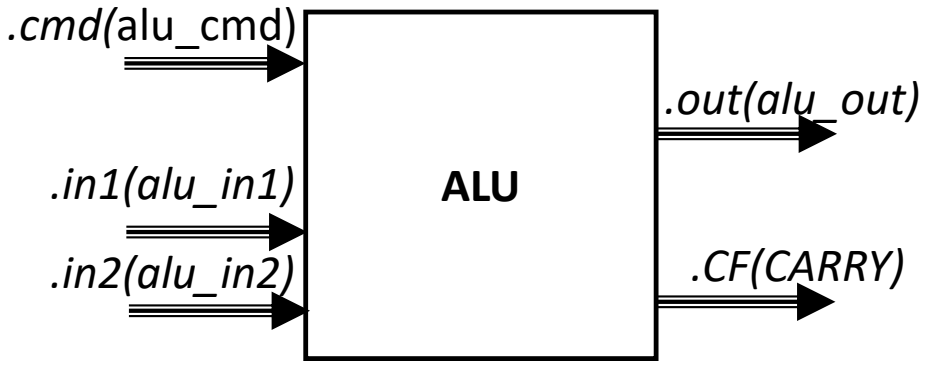


Εντολές μηχανής (instruction set) του MicroCPU

- **Κλασσικές εντολές επεξεργασίας** που εμπλέκουν την ALU για αριθμητικές και λογικές πράξεις, π.χ.:
  AND operand1 operand2 operand3
  OR operand1 operand2 operand3
  XOR operand1 operand2 operand3
  ADD operand1 operand2 operand3
  Το αποτέλεσμα γράφεται στον operand1. Οι πράξεις εμπλέκουν δύο operands τους operand2 και operand3.
  Operands για τον MicroCPU είναι μόνο οι καταχωρητές του.
- **Εντολές μετακίνησης δεδομένων από καταχωρητή σε καταχωρητή**:
  MOV Rd R
  τα δεδομένα αντιγράφονται από τον καταχωρητή R στον καταχωρητή Rd (το Rd προκύπτει από το register destination).
- **Εντολές φόρτωσης δεδομένων από μνήμη σε καταχωρητή**:
  Load_FROM_MEM Rd address
  αντιγράφει τα δεδομένα από την διεύθυνση μνήμης address στον καταχωρητή Rd.
- **Εντολές αποθήκευσης δεδομένων από καταχωρητή στην μνήμη**:
  STORE_TO_MEM address R
  αποθηκεύει στη διεύθυνση μνήμης address τα δεδομένα του καταχωρητή R.
- **Εντολές Αρχικοποίησης τιμών**:
  OP_SHORT_TO_REG Rd value
  αντιγράφει την τιμή value των 8 bits στον καταχωρητή Rd.
- **Εντολές διακλάδωσης:**
  BNZ Rc address
  μετακινεί τον program counter στην τιμή address αν ο καταχωρητής Rc δεν είναι 0. (Branch if Not Zero)

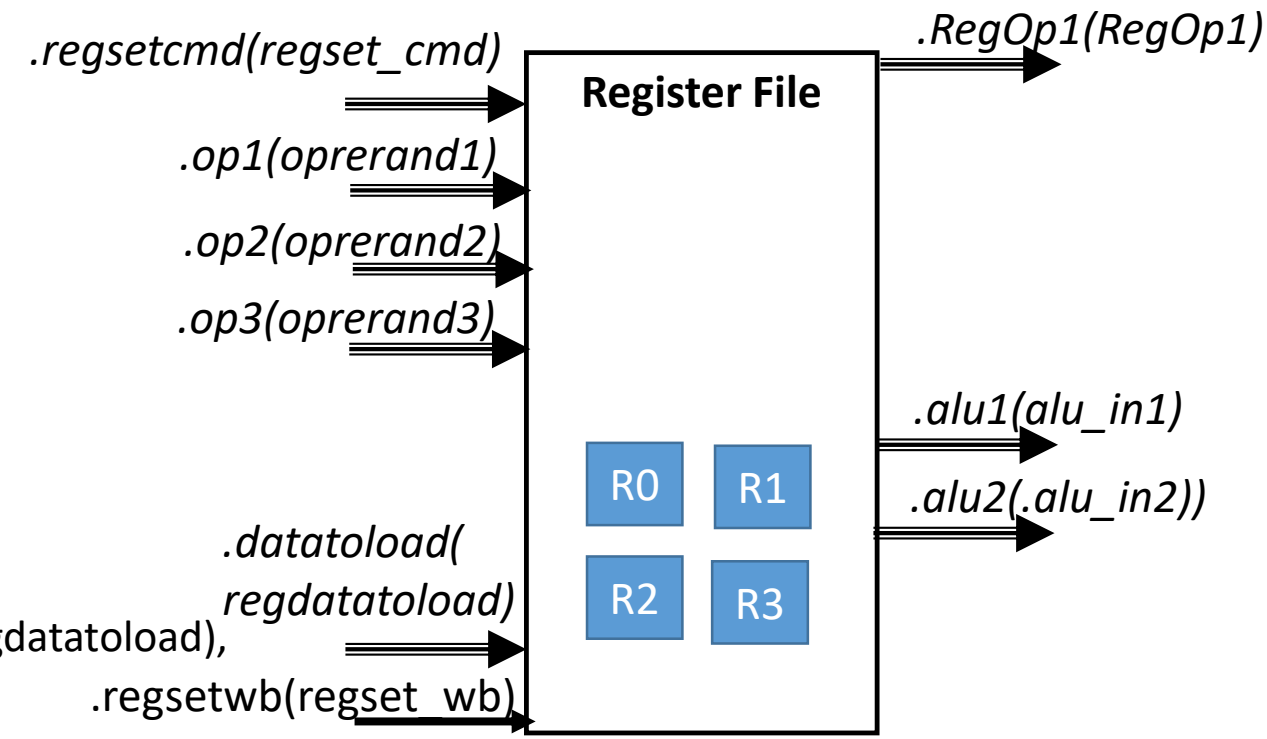
Η μονάδας ελέγχου του MicroCPU – instance της ALU

```
//control signals for ALU
wire [WORD_SIZE-1:0] alu_in1;
wire [WORD_SIZE-1:0] alu_in2;
wire [WORD_SIZE-1:0] alu_out;
wire CARRY;
wire [ALU_CMD_SIZE-1:0] alu_cmd;
MCPU_Alu #(.CMD_SIZE(ALU_CMD_SIZE), .WORD_SIZE(WORD_SIZE))
aluinst (.cmd(alu_cmd),
        .in1(alu_in1),
        .in2(alu_in2),
        .out(alu_out),
        .CF(CARRY));
```



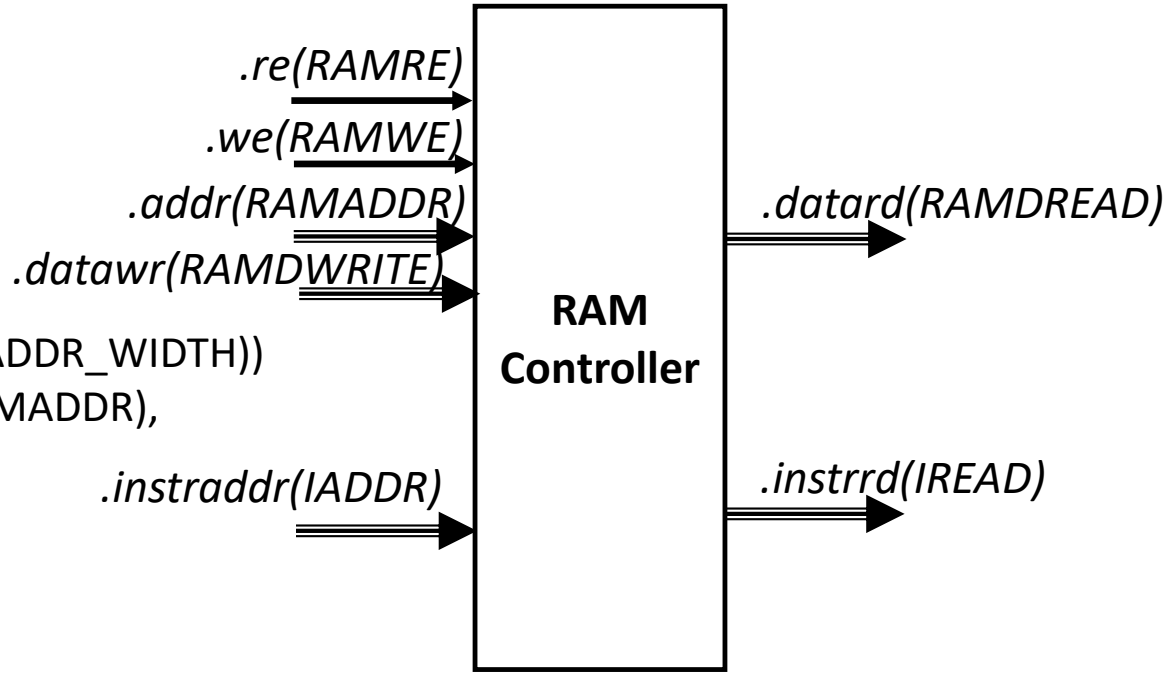
Η μονάδας ελέγχου του MicroCPU – instance του register file

```
//The Program Counter
reg [ADDR_WIDTH-1:0] pc;

//Control Bits for Register file
reg [1:0] regset_cmd;
reg regset_wb;
wire [WORD_SIZE-1:0] regdatatoload;
wire [WORD_SIZE-1:0] RegOp1;

MCPU_Registerfile #(.WORD_SIZE(WORD_SIZE),
.OPERAND_SIZE(OPERAND_SIZE))
regfileinst (.op1(operand1), .op2(operand2), .op3(operand3),
.RegOp1(RegOp1), .alu1(alu_in1), .alu2(alu_in2), .datatoload(regdatatoload),
.regsetwb(regset_wb), .regsetcmd(regset_cmd));
```



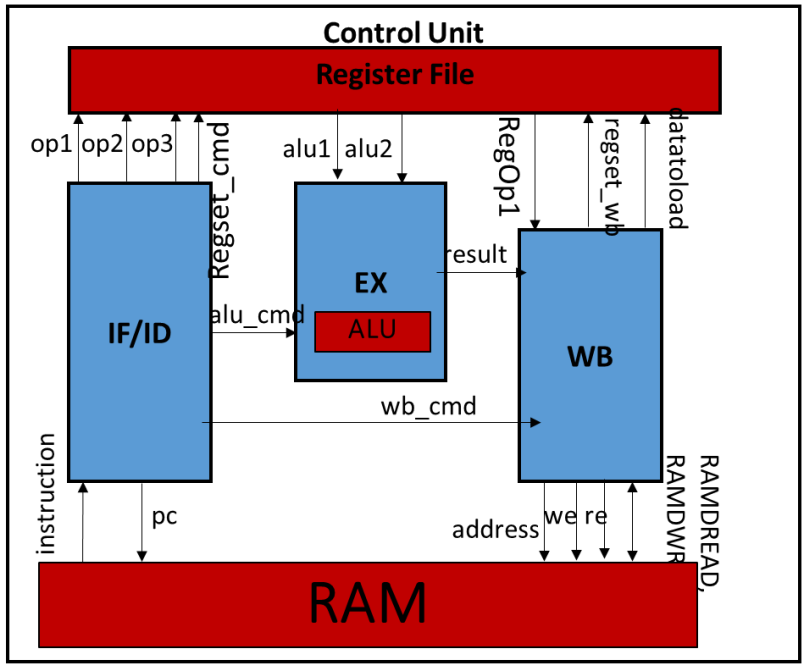
Η μονάδας ελέγχου του MicroCPU – instance της μνήμης

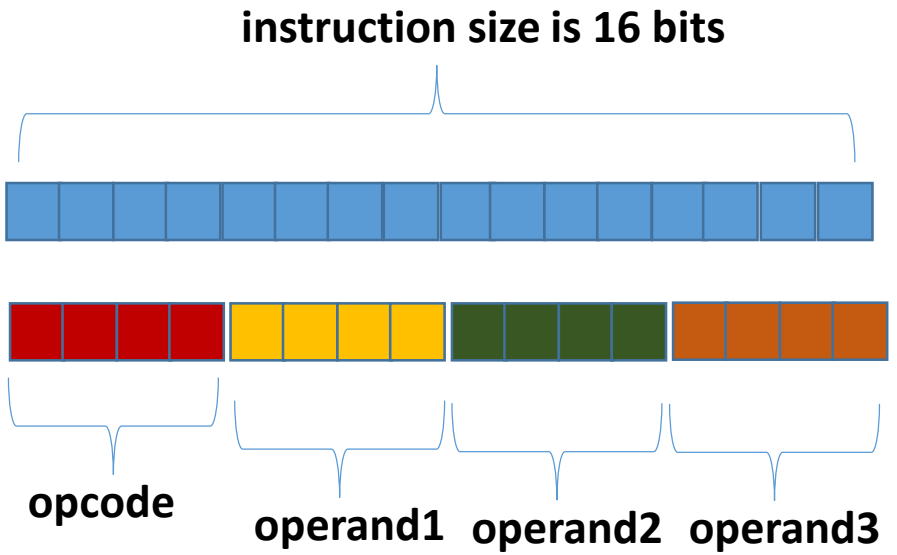
```
//Control Bits for RAM
reg RAMWE, RAMRE;
reg [ADDR_WIDTH-1:0] RAMADDR;
wire [WORD_SIZE-1:0] RAMDWRITE;
wire [WORD_SIZE-1:0] RAMDREAD;

wire [ADDR_WIDTH-1:0] IADDR;
wire [WORD_SIZE-1:0] IREAD;

MCPU_RAMController #(.WORD_SIZE(WORD_SIZE), .ADDR_WIDTH(ADDR_WIDTH))
raminst (.we(RAMWE), .datawr(RAMDWRITE), .re(RAMRE), .addr(RAMADDR),
.datard(RAMDREAD), .instraddr(IADDR), .instrrd(IREAD));
```

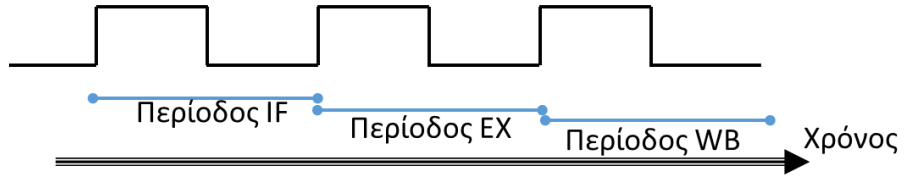


Η μονάδας ελέγχου του MicroCPU – Δομική περιγραφή Συνδυαστικού τμήματος

```
//this is always operand 3 value as a register
assign RAMDWRITE=RegOp1;
assign instruction=IREAD;
assign opcode=instruction[INSTRUCTION_SIZE-1:INSTRUCTION_SIZE-OPCODE_SIZE];
assign aluopcode=opcode[ALU_CMD_SIZE-1:0];
assign operand1=instruction[OPCODE_SIZE*3-1:2*OPCODE_SIZE];
assign operand2=instruction[OPCODE_SIZE*2-1:OPCODE_SIZE];
assign operand3=instruction[OPCODE_SIZE-1:0];
assign IADDR=pc;
wire [WORD_SIZE-1:0] MemOrConstant;
assign datatoread=(opcode==OP_SHORT_TO_REG)?{8'b00000000, operand2, operand3}:RAMDREAD;
assign datatoload=(regset_cmd==regfileinst.NORMAL_EX)?alu_out:dataread;
```

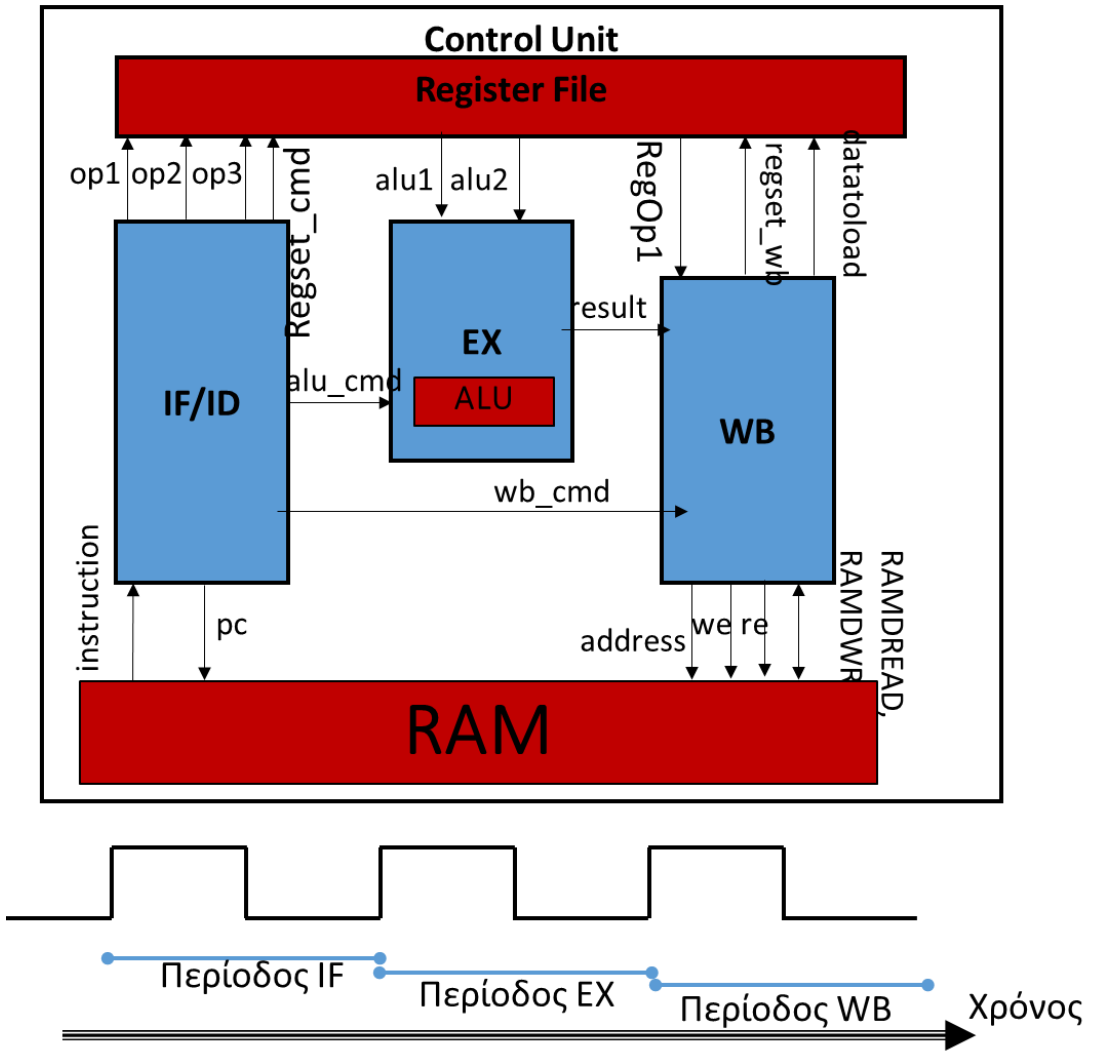






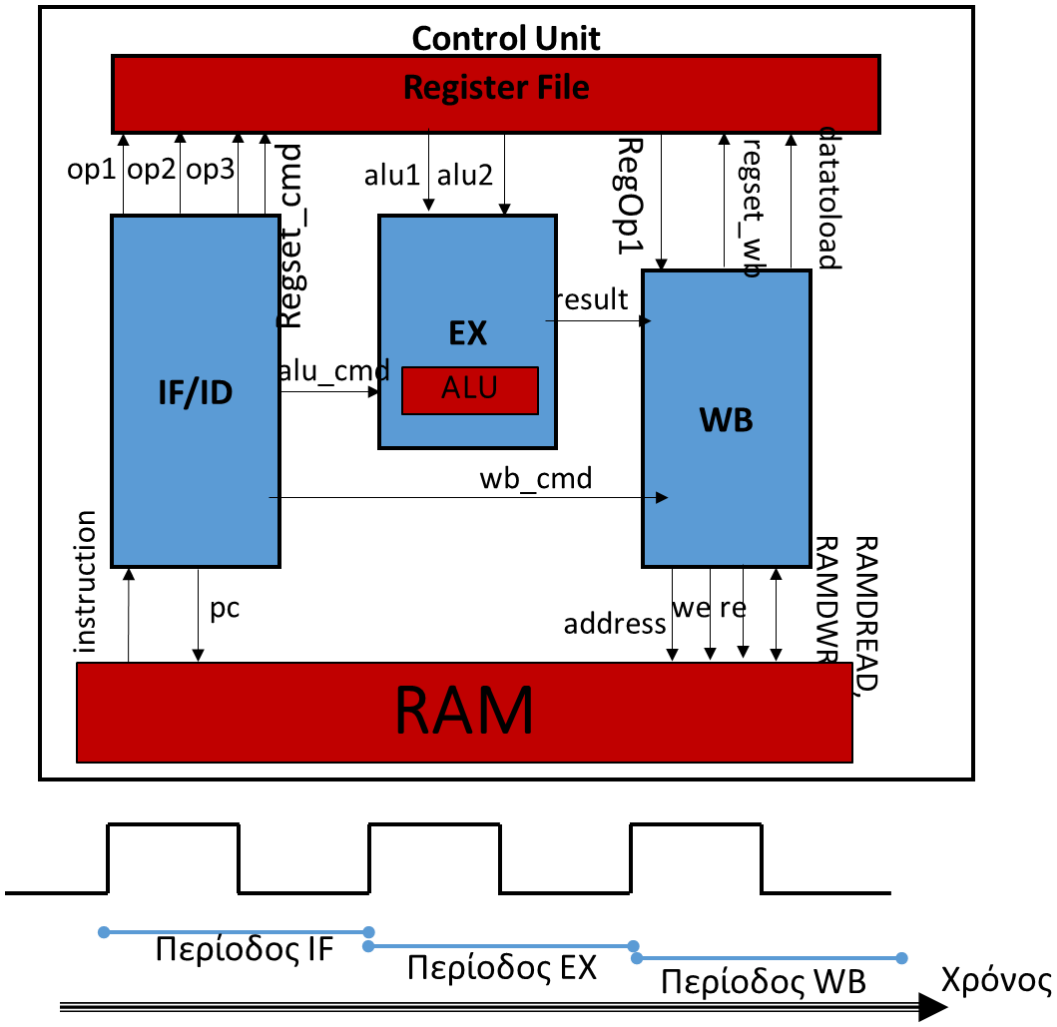
Η μονάδας ελέγχου του MicroCPU – καταστάσεις για το ακολουθιακό τμήμα

```
//parameter CPU_STATES_BITS=2;
//Instruction Fetch State
parameter  [1:0]  IF_STATE  = 2'b00;
//Execute Fetch State
parameter  [1:0]  EX_STATE  = 2'b01;
//WriteBack State
parameter  [1:0]  WB_STATE  = 2'b10;
//HALTED State
parameter  [1:0]  HLT_STATE  = 2'b11;

reg [1:0] state;
reg [1:0] next_state;
```



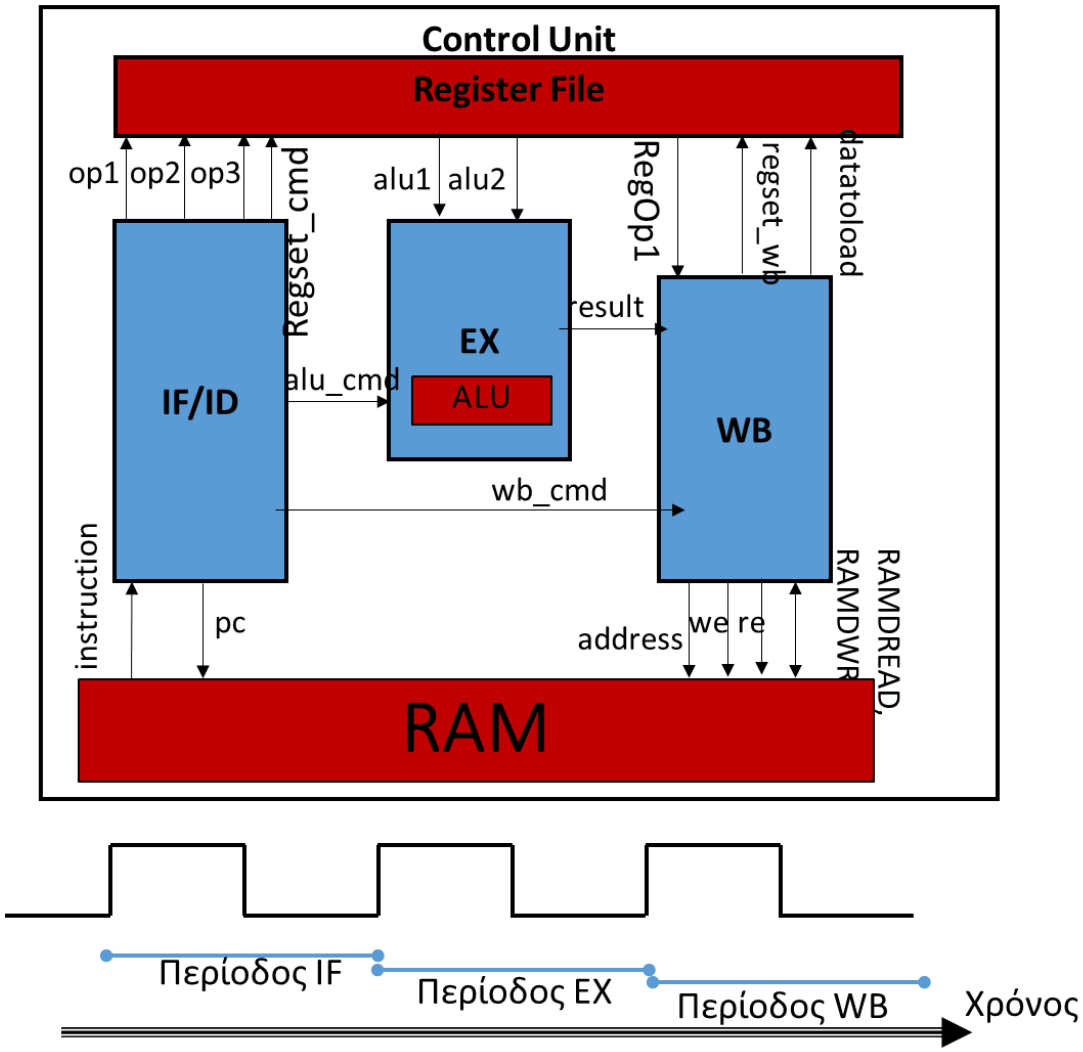
Η μονάδας ελέγχου του MicroCPU - για μετατροπή κατάστασης σε string για debugging purposes

```
reg [8*3:0] STATE_AS_STR;
always @(state)
begin
 case(state)
  IF_STATE:
  begin
   STATE_AS_STR<="IF";
  end
  EX_STATE:
  begin
   STATE_AS_STR<="EX";
  end
  WB_STATE:
  begin
   STATE_AS_STR<="WB";
  end
  default:
  begin
   STATE_AS_STR<="HLT";
  end
 endcase
end
```



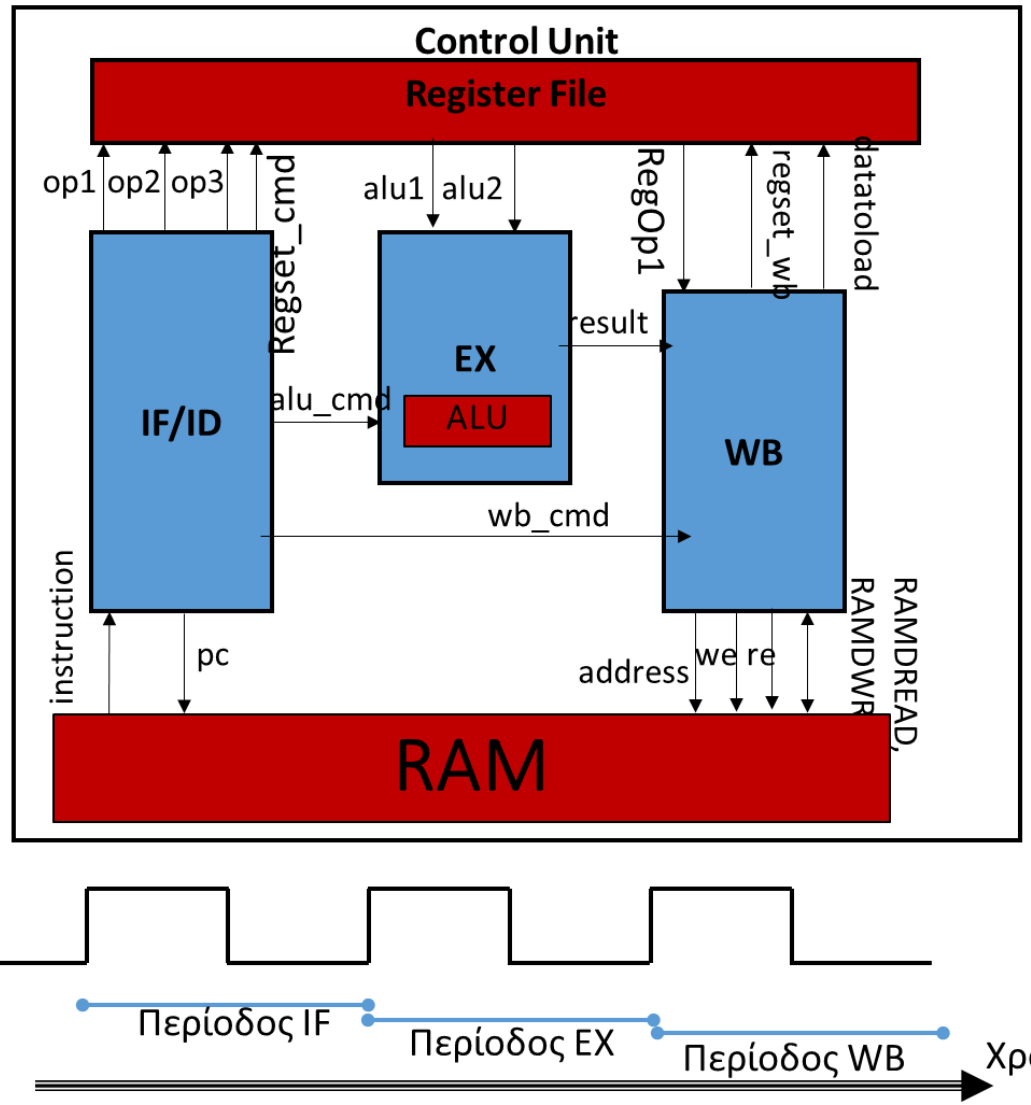
Η μονάδας ελέγχου του MicroCPU – Ασύγχρονο FSM

```
always @ (state, opcode)
begin : MAIN_FSM_ASYCHRONOUS
  next_state=0;
  case(state)
    IF_STATE:
    begin
      //this is a 3 stages pipelining so IF and DECODE occur on this step
      case(opcode)
        OP_BNZ:
        begin
           next_state = IF_STATE;
        end
        default:
        begin
           next_state = EX_STATE;
        end
      endcase
    end
    EX_STATE:
    begin
      next_state = WB_STATE;
    end
    WB_STATE:
    begin
      next_state = IF_STATE;
    end
  endcase
end
```



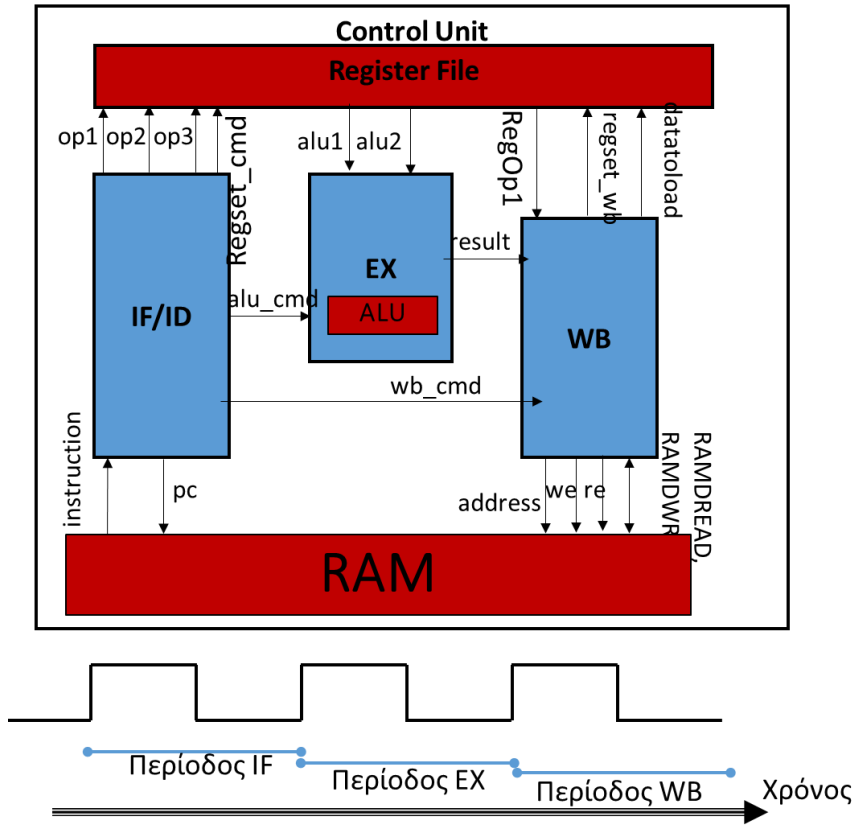
Η μονάδας ελέγχου του MicroCPU – Σύγχρονο FSM

```
always @ (posedge clk, reset)
begin : MAIN_FSM
  if (reset == 1'b1) begin
    //get the CPU into IF state
    state <= #1 IF_STATE;
  //reset the Program Counter PC
    pc <= #1 0;
  end else begin
    case(state)
```



Η μονάδας ελέγχου του MicroCPU – IF/ID state handling

```
 case(state)
 IF_STATE:
begin
//this is a 3 stages pipelining so IF and DECODE occur on this step
  case(opcode)
   OP_AND,OP_OR,OP_XOR,OP_ADD:
   begin
    regset_cmd <= #2 regfileinst.NORMAL_EX;
   end
   OP_MOV:
   begin
    regset_cmd <= #2 regfileinst.MOV_INTERNAL;
   end
   OP_LOAD_FROM_MEM:
   begin
    regset_cmd<= #2 regfileinst.LOAD_FROM_DATA;
    wb_cmd[ADDR_WIDTH-1:0]<= #2 {operand2,operand3};
    wb_cmd[ADDR_WIDTH]<= #2 1'b0; //RAMWE
    wb_cmd[ADDR_WIDTH+1]<= #2 1'b1; //RAMRE
   end
   OP_STORE_TO_MEM:
   begin
    regset_cmd <= #2 regfileinst.DO_NOTHING;
    //whatever is in RAMDWRITE is written to address {operand1,operand2}
    //it is also the content of register pointed by operand 1
    wb_cmd[ADDR_WIDTH-1:0] <= #2 {operand1,operand2};

    wb_cmd[ADDR_WIDTH] <= #2 1'b1; //RAMWE - RAM WRITE ENABLE
    wb_cmd[ADDR_WIDTH+1] <= #2  1'b0; //RAMRE - RAM READ ENABLE
   end
   OP_SHORT_TO_REG:
   begin
    regset_cmd<=#2 regfileinst.LOAD_FROM_DATA;
    wb_cmd[ADDR_WIDTH]<=#2 1'b0; //RAMWE
    wb_cmd[ADDR_WIDTH+1]<=#2 1'b0; //RAMRE
   end
   OP_BNZ:
   begin
    if(RegOp1!=0)
    begin
     pc <= #5 {operand2, operand3};
    end else
    begin
     pc <= #5 pc+1;
    end
   end
   default:
   begin
   end
  endcase
end
```



Η μονάδας ελέγχου του MicroCPU – EX και WB states handling

```
EX_STATE:
  begin
  end
WB_STATE:
  begin
    RAMADDR<=#1 wb_cmd[ADDR_WIDTH-1:0];
    RAMWE<=#1 wb_cmd[ADDR_WIDTH];
    RAMRE<=#1 wb_cmd[ADDR_WIDTH+1];

    regset_wb <= #2 1'b1;
    regset_wb<= #3 1'b0;
    wb_cmd[ADDR_WIDTH]<= #3 1'b0;
    wb_cmd[ADDR_WIDTH+1]<=  #3 1'b0;
    RAMWE <= #3 1'b0;
    RAMRE <= #3 1'b0;
    pc <= #5 pc+1;
  end
  HLT_STATE:
  begin
    $display("processor HALTED\n");
    $stop;
  end
  default:
  begin
  end
```