

Γλώσσες Περιγραφής Υλικού I

Προγραμματιζόμενοι επεξεργαστές

Νικόλαος Καββαδίας
nkavn@uop.gr

01 Ιουνίου 2011

- Προγραμματιζόμενοι επεξεργαστές
- Ρεαλιστικό παράδειγμα: ο επεξεργαστής ΜU0 (MicroProcessor ZERO)
 - Το σύνολο εντολών του ΜU0
 - Η μικροαρχιτεκτονική οργάνωση του ΜU0
 - Εκτέλεση προγράμματος στον ΜU0
 - Παραδείγματα απλών εφαρμογών

Προγραμματιζόμενοι επεξεργαστές

- Προγραμματιζόμενοι επεξεργαστές είναι εκείνα τα κυκλώματα τα οποία έχουν σχεδιαστεί έτσι ώστε να μπορούν οποιοδήποτε πρόβλημα μπορεί να περιγραφεί στο σύνολο εντολών τους
- Κύρια χαρακτηριστικά της αρχιτεκτονικής ενός επεξεργαστή
 - Σύνολο (ρεπερτόριο) εντολών
 - Κωδικοποίηση των εντολών (ορθογωνιότητα, μήκος εντολής)
 - Σύνταξη (κειμενική διαμόρφωση με χρήση μνημονικών για διευκόλυνση του προγραμματιστή)
 - Μικροαρχιτεκτονική του επεξεργαστή
 - Μνήμη προγράμματος, δεδομένων, καταχωρητές, λειτουργικές μονάδες (datapath)
 - Μονάδα ελέγχου: μικροπρογραμματιζόμενη ή FSM
 - Εκτέλεση των εντολών στο χρόνο (οργάνωση μονού/πολλαπλών κύκλων, με/χωρίς διοχέτευση)
 - Περιγραφή σε σχηματική μορφή (schematics) ή
 - Περιγραφή σε κάποια HDL (VHDL, Verilog HDL)

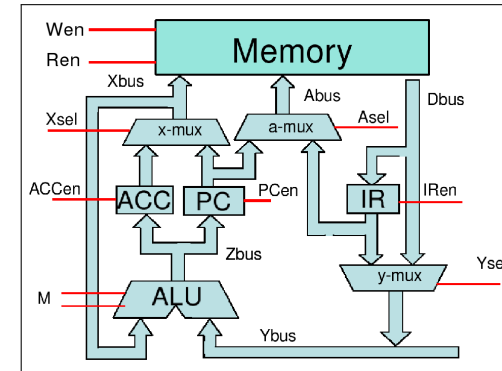
Γενικά χαρακτηριστικά του επεξεργαστή ΜU0

- Μικροεπεξεργαστής με αρχιτεκτονική Von Neumann
 - Κοινός χώρος διευθύνσεων για εντολές και δεδομένα (στην ίδια φυσική μνήμη)
- Διαθέτει εντολές και δεδομένα των 16-bit
- Πεδίο διευθύνσεων των 12-bit ($2^{12} = 4096$ εντολές των 16-bit)
- Κάθε εντολή χρειάζεται δύο κύκλους μηχανής για να εκτελεστεί
- Κάθε εντολή αποτελείται από δύο φάσεις:
 - fetch (φόρτωση)
 - decode and execute (αποκωδικοποίηση και εκτέλεση)
- Διαμόρφωση εντολής: κωδικός λειτουργίας (opcode) των 4-bit και γενικό πεδίο των 12-bit
- Λειτουργία καταχωρητών στη θετική ακμή και μνήμης στην αρνητική ακμή ρολογιού

Το σύνολο εντολών του MU0

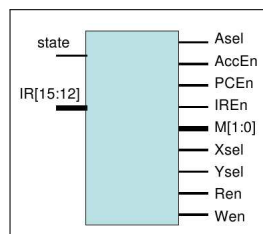
Instruction	Opcode	Effect
LDA S	0000	ACC := mem ₁₆ [S]
STO S	0001	mem ₁₆ [S] := ACC
ADD S	0010	ACC := ACC + mem ₁₆ [S]
SUB S	0011	ACC := ACC - mem ₁₆ [S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC != 0 PC := S
STP	0111	stop

Ο χειριστής δεδομένων (datapath) του MU0



- Αριθμητική-λογική μονάδα: alu16.v
- Πολυπλέκτες δεδομένων x-mux, y-mux: mux16.v
- Πολυπλέκτης διεύθυνσης a-mux: mux12.v
- PC (Program Counter) και IR (Instruction Register): vrea16.v

Η μονάδα ελέγχου (controller) του MU0



- Η μονάδα ελέγχου έχει δύο εισόδους:
 - state: τρέχουσα κατάσταση του FSM
 - IR[15:12]: ο κωδικός λειτουργίας της τρέχουσας εντολής
- Οι έξοδοι της μονάδας ελέγχου ελέγχουν και ενεργοποιούν τις υπομονάδες του χειριστή δεδομένων
- Λειτουργία
 - 1) Ανάκληση της επόμενης εντολής από τη μνήμη και αποθήκευσή της στον IR
 - 2) Αποκωδικοποίηση και εκτέλεση της εντολής

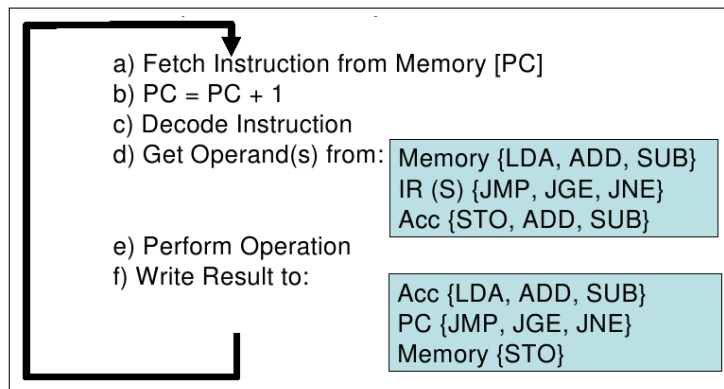
Πίνακας καταστάσεων του FSM

state	F[2:0]	Next state	IREn	PCEn	AccEn	M[1:0]	Xsel	Ysel	Asel	Ren	Wen
0	xxx	1	1	1	0	10	0	x	0	1	0
1	000	0	0	0	1	00	0	1	1	1	0
1	001	0	0	0	0	xx	1	x	1	0	1
1	010	0	0	0	1	01	1	1	1	1	0
1	011	0	0	0	1	11	1	1	1	1	0
1	100	0	0	1	0	00	0	0	x	0	0
1	101	0	0	N	0	00	0	0	x	0	0
1	110	0	0	Z	0	00	0	0	x	0	0
1	111	1	0	0	0	xx	0	x	x	0	0

Notes:

- N and Z are the Negative and Zero state of the Accumulator, respectively. (used to reduce the size of the table, as drawn)
- If a value is not going to be latched it doesn't matter what it is! (e.g. ALU output for STO)
- STP operates by remaining in its evaluation state.

Ο γενικός κύκλος εκτέλεσης εντολών στον ΜU0



Ο ΜU0 σε δράση: Κατάσταση FETCH

- Ο κύκλος FETCH συμβαίνει στην πρώτη θετική ακμή του ρολογιού
- Η είσοδος του a-mux συνδέεται με την έξοδο του PC ο οποίος δείχνει στην επόμενη εντολή στη μνήμη
- Στην επόμενη αρνητική ακμή, μεταφέρονται τα περιεχόμενα εκείνης της θέσης από τη μνήμη στο δίαυλο Dbus
- Τα περιεχόμενα του Dbus βρίσκονται στην είσοδο του IR
- Τα περιεχόμενα του PC βρίσκονται και στην είσοδο της ALU μέσω του πολυπλέκτη x-mux. Η είσοδος ελέγχου M της ALU έχει τέτοια τιμή ώστε η ALU να αυξήσει την τιμή που έχει στην είσοδό της. Στην επόμενη θετική ακμή τα περιεχόμενα του PC θα έχουν αυξηθεί ώστε να είναι έτοιμα για την επόμενη φορά που θα βρεθεί η FSM σε κατάσταση FETCH

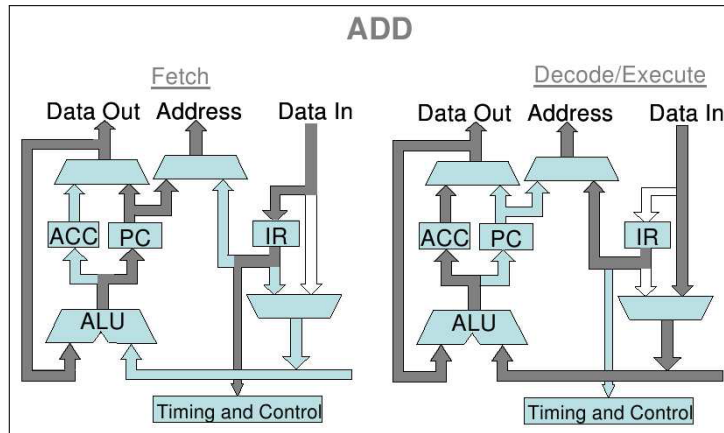
Ο ΜU0 σε δράση: Κατάσταση EXEC (1)

- Ο κύκλος EXEC συμβαίνει στην επόμενη θετική ακμή του ρολογιού
- Ο PC αυξάνει τα περιεχόμενά του
- Ο IR αποθηκεύει τα περιεχόμενα του Dbus
- Η μονάδα ελέγχου διαβάσει τα ψηφία [15:12] από τον IR (κωδικός λειτουργίας)
- Ανάλογα με την τιμή του κωδικού λειτουργίας, ενεργοποιούνται ή απενεργοποιούνται συγκεκριμένες υπομονάδες του χειριστή δεδομένων
 - opcode = LDA
 - ενεργοποιείται ο ACC και ο Dbus συνδέεται με την είσοδο της ALU στο δίαυλο Ybus. Η τιμή αυτή διέρχεται απλώς από την ALU. Στην επόμενη θετική ακμή, η έξοδος του ACC αποθηκεύεται στο Dbus

Ο ΜU0 σε δράση: Κατάσταση EXEC (2)

- (συνέχεια)
 - opcode = ADD, SUB
 - Ενεργοποιείται ACC και ο Dbus συνδέεται με την ALU μέσω του y-mux. Ο x-mux επιτρέπει τα δεδομένα στο Xbus και η τιμή του M είναι ADD ή SUB. Στην επόμενη αρνητική ακμή, τα περιεχόμενα της μνήμης μεταφέρονται στο Dbus. Στην επόμενη θετική ακμή, η έξοδος του ACC αποθηκεύει το άθροισμα της προηγούμενης τιμής του με αυτό που μεταφέρθηκε από τη μνήμη
 - opcode = STO
 - Ο x-mux τοποθετεί τα περιεχόμενα του ACC στο Xbus. Ο IR[11:0] μεταφέρεται μέσω του a-mux στο δίαυλο διεύθυνσης της μνήμης. Στην επόμενη αρνητική ακμή, η μνήμη αποθηκεύει τα περιεχόμενα του Xbus
 - opcode = JMP, JGE, JNE
 - Ο IR[11:0] μεταφέρεται μέσω του y-mux στο Ybus και στην ALU. Η τιμή αυτή διέρχεται από την ALU ώστε να γραφεί στον PC. Στην επόμενη θετική ακμή (FETCH) η τιμή του PC είναι πλέον αυτή που προέκυψε από την εντολή άλματος

Ο κύκλος εκτέλεσης της εντολής ADD



Το αρχείο γενικών παραμέτρων (defs.h)

- Συγγραφέας των αρχικών μοντέλων Verilog: Gerard Borg
- Ιστοσελίδα:
<http://engnet.anu.edu.au/DEpeople/Gerard.Borg/>
- Διορθώσεις από τον υποφαινόμενο και νέα αρχεία:
mu0_controller.v, mu0_datapath.v, mu0.v, mu0_tb.v

```
/**
 * project MU0
 * module defs, some parameters for MU0
 * G. Borg
 * Feb 2009
 **/
parameter MAXDEPTH = 12;
parameter MAXWIDTH = 16;
```

Η αριθμητική-λογική μονάδα (alu16.v)

```
module alu16(M, X, Y, Z);
`include "defs.h"
parameter Acc = 00; //Pass through to the Accumulator
parameter add = 01; //Add X and Y
parameter PCi = 10; //Increment the Program counter
parameter sub = 11; //Subtract Y from X
input [1:0] M;
input [MAXWIDTH-1:0] X;
input [MAXWIDTH-1:0] Y;
output [MAXWIDTH-1:0] Z;
reg [MAXWIDTH-1:0] Z;

always @(M, X, Y)
begin
case(M)
add: Z = X + Y;
sub: Z = X - Y + 1;
PCi: Z = X + 1;
Acc: Z = Y;
default: Z = 0;
endcase
end
endmodule //alu16.v
```

Τα αρχεία πολυπλεκτών (mux16.v, mux12.v)

Πολυπλέκτης 16-bit

```
module mux16(z, s, a, b);
`include "defs.h"
output [MAXWIDTH-1:0] z;
input [MAXWIDTH-1:0] a, b;
input s;

assign z = s ? b : a; // s=1, z<-b; s=0, z<-a
endmodule //mux16.v
```

Πολυπλέκτης 12-bit

```
module mux12(z, s, a, b);
`include "defs.h"
output [MAXDEPTH-1:0] z;
input [MAXDEPTH-1:0] a, b;
input s;

assign z = s ? b : a; // s=1, z<-b; s=0, z<-a
endmodule //mux12.v
```

Καταχωρητής PC και IR (vreg16.v)

```
module vreg16 (clk, q, d, en, rs);
`include "defs.h"
output [MAXWIDTH-1:0] q;
input [MAXWIDTH-1:0] d;
input clk;
input en;
input rs;
reg [MAXWIDTH-1:0] q;

always @(posedge clk)
begin
if (en && ~rs)
q <= d;
if (en && rs)
q <= 16'h0;
end
endmodule // v_reg16
```

Απλό, μη-συνθέσιμο μοντέλο μνήμης (mem.v)

```
module mem (Clk, Wen, Ren, address, write_data, read_data);
`include "defs.h"
input Clk;
input Wen;
input Ren;
input [MAXDEPTH-1:0] address;
input [MAXWIDTH-1:0] write_data;
output [MAXWIDTH-1:0] read_data;
reg [MAXWIDTH-1:0] read_data;
reg [MAXWIDTH-1:0] mem [12'h0:12'hFFF];

initial
$readmemh("prog.lst", mem);

always @(negedge Clk)
begin
if (Wen)
mem[address] <= write_data;
if (Ren)
read_data <= mem[address];
end
endmodule //mem.v
```

Ο χειριστής δεδομένων του MU0 (mu0_datapath.v) (1)

```
module mu0_datapath(clk, reset,
Abus, Xbus, Dbus, acc, ir, pc,
M, PCen, Iren, ACCen, Xsel, Ysel, Asel);
`include "defs.h"
input clk, reset;
input Xsel, Ysel, Asel, ACCen, PCen, Iren;
input [1:0] M;
output [MAXWIDTH-1:0] acc;
output [MAXWIDTH-1:0] ir;
output [MAXWIDTH-1:0] pc;
output [MAXDEPTH-1:0] Abus; // Address bus
output [MAXWIDTH-1:0] Xbus;
output [MAXWIDTH-1:0] Dbus; // Data input to MU0
wire [MAXWIDTH-1:0] Ybus;
wire [MAXWIDTH-1:0] Zbus; // ALU output
```

Ο χειριστής δεδομένων του MU0 (mu0_datapath.v) (2)

```
alu16 alu (
.M(M), .X(Xbus), .Y(Ybus), .Z(Zbus)
);

mux16 x_mux (
.z(Xbus), .s(Xsel), .a(pc), .b(acc)
);
mux16 y_mux (
.z(Ybus), .s(Ysel), .a(ir), .b(Dbus)
);
mux12 a_mux (
.z(Abus), .s(Asel),
.a(pc[MAXDEPTH-1:0]), .b(ir[MAXDEPTH-1:0])
);

vreg16 ProgramCounter (
.clk(clk), .q(pc), .d(Zbus), .en(PCen), .rs(reset)
);
vreg16 InstructionRegister (
.clk(clk), .q(ir), .d(Dbus), .en(Iren), .rs(reset)
);
vreg16 Accumulator (
.clk(clk), .q(acc), .d(Zbus), .en(ACCen), .rs(reset)
);

endmodule //mu0_datapath
```

Ο επεξεργαστής MU0 (mu0.v)

```
module mu0(clk, reset, pc, ir, acc);
`include "defs.h"
input clk, reset;
wire Asel, Xsel, Ysel, ACCen, PCen, IRen, Ren, Wen;
wire [1:0] M;
output [MAXWIDTH-1:0] pc, ir, acc, Zbus;
wire [MAXDEPTH-1:0] Abus;
wire [MAXWIDTH-1:0] Xbus, Dbus;

// Instruction and data memory (non-synthesizable)
mem memory (
    .Clk(clk), .Wen(Wen), .Ren(Ren),
    .address(Abus), .write_data(Xbus), .read_data(Dbus));

// Control unit
mu0_controller control_unit (
    .clk(clk), .reset(reset), .opcode(ir[MAXWIDTH-1:MAXWIDTH-4]),
    .acc(acc), .Asel(Asel), .ACCen(ACCen), .PCen(PCen), .IRen(IRen),
    .M(M), .Xsel(Xsel), .Ysel(Ysel), .Ren(Ren), .Wen(Wen));

// Datapath
mu0_datapath datapath (
    .clk(clk), .reset(reset),
    .Abus(Abus), .Xbus(Xbus), .Dbus(Dbus), .acc(acc), .ir(ir), .pc(pc),
    .M(M), .PCen(PCen), .IRen(IRen), .ACCen(ACCen),
    .Xsel(Xsel), .Ysel(Ysel), .Asel(Asel));
endmodule //mu0
```

Νικόλαος Καββαδίας nkavn@uop.gr

Γλώσσες Περιγραφής Υλικού I

Έλεγχος ορθής λειτουργίας με testbench (mu0_tb.v)

```
`timescale 1 ns / 10 ps
module mu0_tb;
`include "defs.h"
reg clk, reset;
wire [MAXWIDTH-1:0] pc, ir, acc;

mu0 processor (
    .clk(clk), .reset(reset), .pc(pc), .ir(ir), .acc(acc)
);

initial begin
    clk = 1'b0; reset = 1'b1;
    #10 reset = 1'b0;
end
always
    #5 clk = ~clk;

initial begin
    $dumpfile("mu0.vcd");
    $dumpvars(0, mu0_tb);
end

initial begin
    #1000;
    $finish;
end
endmodule
```

Νικόλαος Καββαδίας nkavn@uop.gr

Γλώσσες Περιγραφής Υλικού I

Προσομοίωση του MU0 για απλό πρόγραμμα δοκιμής (1)

- Το εκάστοτε πρόγραμμα δοκιμής δηλώνεται ως αρχείο 'prog.lst', τα δεδομένα του οποίου φορτώνονται μέσω της \$readmemh
- Η φόρτωση γίνεται στο mem.v
- Απλό πρόγραμμα

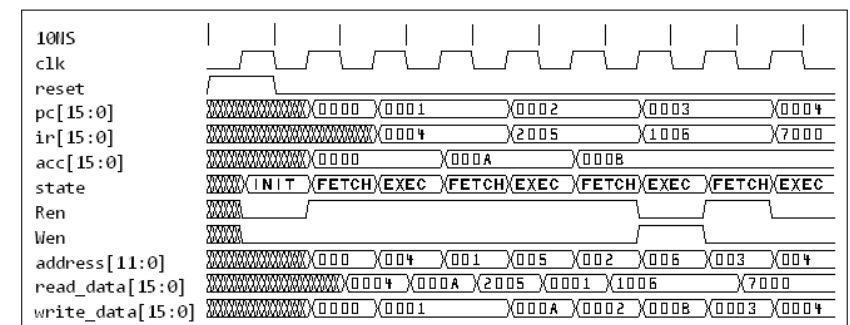
```
0004 // load (LDA) the contents of memory address 4 into the ACC
2005 // add (ADD) the contents of memory address 5 to that in the ACC
1006 // store (STO) the contents of the ACC in memory location 6
7000 // STOP
000A // data stored in memory location 4
0001 // data stored in memory location 5
0000 // data stored in memory location 6
```

Νικόλαος Καββαδίας nkavn@uop.gr

Γλώσσες Περιγραφής Υλικού I

Προσομοίωση του MU0 για απλό πρόγραμμα δοκιμής (2)

Χρονικό διάγραμμα (τιμές στο δεκαεξαδικό)



Νικόλαος Καββαδίας nkavn@uop.gr

Γλώσσες Περιγραφής Υλικού I

Μη συνθέσιμο, behavioral μοντέλο του ΜU0 (mu0_behav.v) (1)

```
module mu0(clk, reset, pc, ir, acc);
  parameter MAXWIDTH = 16, MAXDEPTH = 12;
  parameter [3:0] LDA = 4'b0000, STO = 4'b0001, ADD = 4'b0010, SUB = 4'b0011;
  parameter [3:0] JMP = 4'b0100, JGE = 4'b0101, JNE = 4'b0110, STP = 4'b0111;
  input clk, reset;
  output [MAXWIDTH-1:0] pc, ir, acc;
  reg [MAXWIDTH-1:0] pc, ir, acc;
  reg [MAXWIDTH-1:MAXWIDTH-4] opcode;
  reg [MAXWIDTH-5:0] address;
  reg [MAXWIDTH-1:0] mem [12'h0:12'hFFF];

  initial
    $readmemh("prog.lst", mem);

  always @(posedge clk)
  begin
    if (reset == 1'b1) begin
      ir <= 0;
      pc <= 0;
      acc <= 0;
    end
    else begin
      ir = mem[pc];
      if (ir[MAXWIDTH-1:MAXWIDTH-4] != STP) begin
        pc = pc + 1;
      end
    end
  end
end
```

Μη συνθέσιμο, behavioral μοντέλο του ΜU0 (mu0_behav.v) (2)

```
always @(opcode or address or ir or pc or acc)
begin
  opcode = ir[MAXWIDTH-1:MAXWIDTH-4];
  address = ir[MAXWIDTH-5:0];
  case (opcode)
    LDA: acc = mem[address];
    STO: mem[address] = acc;
    ADD: acc = acc + mem[address];
    SUB: acc = acc - mem[address];
    JMP: if (opcode == JMP) begin
      pc = address;
    end
    JGE: if (acc[MAXWIDTH-1] == 1'b0) begin
      pc = address;
    end
    JNE: if (acc != 0) begin
      pc = address;
    end
    default: begin
      end
  endcase
end
endmodule //mu0
```

Ερωτήματα και παρατηρήσεις

- Ποιες οι διαφορές των δύο μοντέλων
 - στο στυλ σύνταξης κώδικα
 - στο μέγεθος κώδικα
 - στη δυσκολία ανάπτυξης
 - στο χρονισμό τους σε κύκλους ρολογιού
 - στον τρόπο χρήσης του μοντέλου μνήμης δεδομένων/προγράμματος
- Πότε προτιμάται η μία και πότε η άλλη τεχνική σχεδίασης
- ☞ Συνήθως η μνήμη δεδομένων/προγράμματος δεν αποτελεί τμήμα του επεξεργαστή