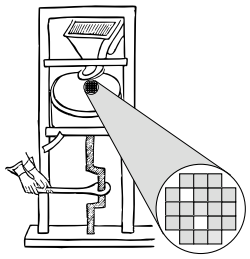
Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch
Holger Lange
Mathias Halbach (FG Rechnerarchitektur)

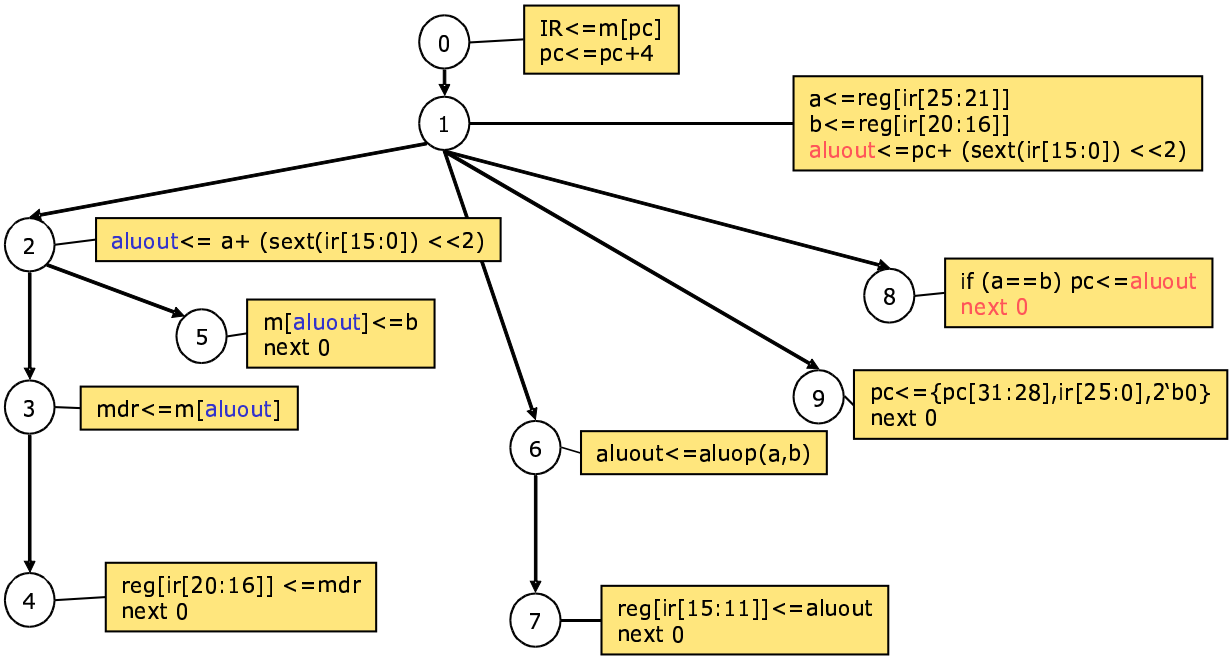


29.06.2006

Technische Grundlagen der Informatik II
8. Übung – MIPS Operationswerk
Sommersemester 2006

Aufgabe 1: MIPS-Mikrooperationen

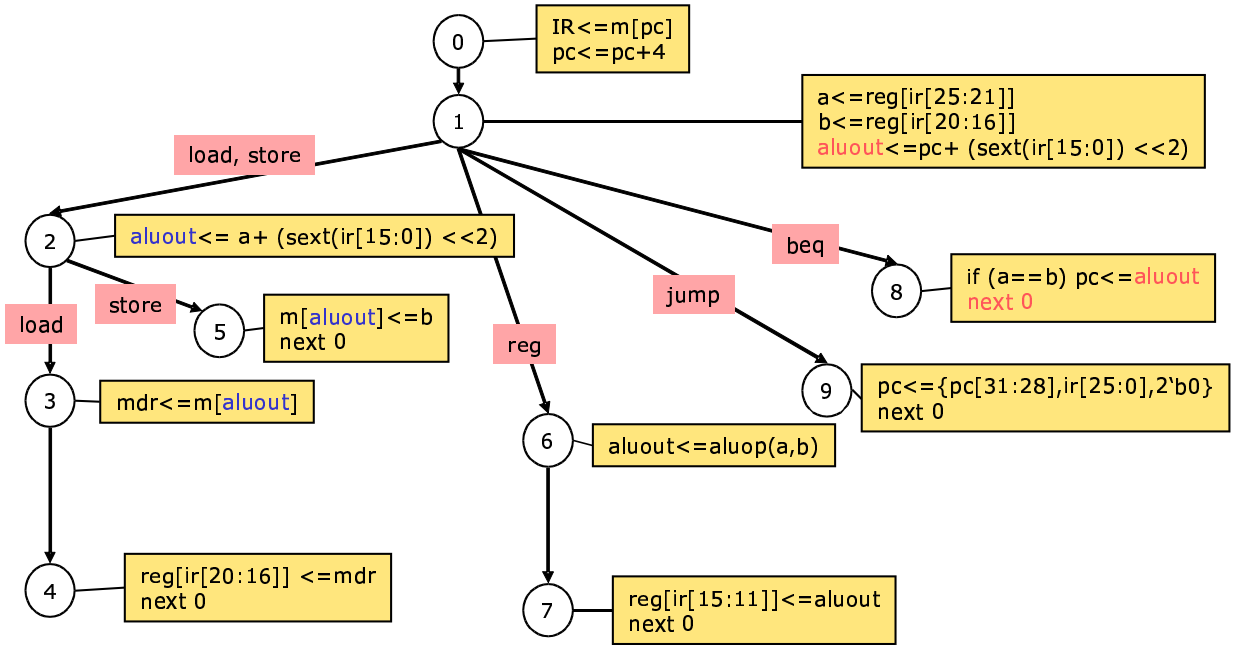
Gegeben ist das folgende Zustandsdiagramm der Mehrtaktimplementierung zur Interpretation der MIPS-Befehle.



a) Ordnen Sie den Verzweigungen jeweils den Befehl oder die Befehlsgruppe zu, in die an den Zuständen 1 und 2 im Zustandsdiagramm verzweigt wird.

Lösung:

Einzutragen sind: **`load`**, **`store`**, **`reg`**, **`jump`** und **`beq`**.



b) Erstellen Sie aus dem Zustandsdiagramm für den aktuellen Befehl an der Speicherstelle m[128] mit dem Befehlsformat 0x01093820 (**`add $7,$8,$9`**) und den Registerwerten **`reg[7]=1`**, **`reg[8]=2`**, **`reg[9]=3`** eine Tabelle, die für jeden Taktschritt (beginnend bei Zustand 0) den jeweiligen Zustand, die auszuführenden Mikrooperationen und die neuen Werte der Register bzw. Speicherzellen enthält.

Lösung:

Zustand	Mikrooperation	Werte der Register und Speicherzellen
0	IR ← m[pc] pc ← pc + 4	
1	a ← reg[ir[25:21]] b ← reg[ir[20:16]] aluout ← pc + (sext(ir[15:0]) << 2)	IR = m[128] = 0x01093820 pc = 128 + 4 = 132
6	aluout ← aluop(a, b)	a = reg[8] = 2 b = reg[9] = 3 aluout = pc + 0x00003820 << 2 = 0xE104
7	reg[ir[15:11]] ← aluout next 0	aluout = 2 + 3 = 5
0		reg[7] = 5

Aufgabe 2: MIPS-Registerspeicher

Implementieren Sie den Registerspeicher (Vorlesung Kapitel 7, Folien 11-13) in Verilog. Nehmen Sie dazu Folie 12 als Vorlage für die Ausgangsbeschaltung, Folie 13 für die Eingangsbeschaltung. Fügen sie ein asynchrones RESET-Signal hinzu.

Lösung:

```
module register_array(CLK, RESET, READ_REGNUM_1, READ_REGNUM_2,
                      WRITE_REGNUM, WRITE_DATA, WRITE,
                      READ_DATA_1, READ_DATA_2);

  parameter log2_n      = 5;
  parameter data_width  = 32;
  parameter n           = 1 << log2_n;

  input                    CLK, RESET, WRITE;
  input  [log2_n-1:0]      READ_REGNUM_1, READ_REGNUM_2, WRITE_REGNUM;
  input  [data_width-1:0]  WRITE_DATA;
  output [data_width-1:0]  READ_DATA_1, READ_DATA_2;

  reg [data_width-1:0] registers [n-1:0];

  assign READ_DATA_1 = registers[READ_REGNUM_1];
  assign READ_DATA_2 = registers[READ_REGNUM_2];

  wire [n-1:0] WRITE_REGNUM_DECODE = 1 << WRITE_REGNUM;

  always @(posedge CLK or posedge RESET) begin: synchronous
    integer i;

    if (RESET) begin
      for (i = 0; i < n; i = i + 1)
        registers[i] <= 0;
    end
    else begin
      for (i = 0; i < n; i = i + 1)
        if ((WRITE_REGNUM_DECODE[i] == 1'b1) && WRITE)
          registers[i] <= WRITE_DATA;
    end
  end

endmodule
```

Aufgabe 3: Fakultät

Entwickeln Sie ein MIPS-Assemblerprogramm zur Berechnung der Fakultät der Zahl 7. Ihr Programm soll den Startwert „7" in das Argumentregister **`a0`** laden, wo er nach dem Programmlauf immer noch stehen soll. Das Ergebnis soll in Register **`v0`** abgelegt werden.

a) Beschreiben Sie den Algorithmus in Pseudo-Code. Verwenden Sie dabei nur Operationen, die im MIPS-Befehlssatz darstellbar sind.

Lösung:

- Lade "7" in Register **`a0`**
- Lade "1" in Register **`v0`** (beim ersten Mal mit eins multiplizieren)
- Kopiere Register **`a0`** nach **`t0`** (**`t0`** ist Schleifenzähler)

- loop: Ist Register `t0` < 2? Ja: Ende, sonst (Ergebnis der Bedingung in `t1`)
- Register `Hi, Lo` = `v0` * `t0`
- Register `v0` = `Lo` (Ergebnis passt in 32 Bit)
- Dekrementiere Register `t0`
- Springe nach Label ”loop”

b) Entwickeln Sie nun aus a) ein MIPS-Assemblerprogramm. Benutzen Sie dazu die in den Vorlesungfolien vorgestellten MIPS-Assemblerbefehle. Verwenden Sie den `addi`-Befehl und das `0`- bzw. `zero`-Register, um Konstanten in Register zu laden.

Lösung:

```
main:   addi    $a0, $0, 7

init:   addi    $v0, $0, 1
        addi    $t0, $a0, 0

loop:   slti    $t1, $t0, 2
        bne     $t1, $0, end

        multu   $v0, $t0
        mflo    $v0
        addi    $t0, $t0, -1
        j       loop

end:    jr      $ra
```