Latvijas Universitāte
Datorikas fakultāte

Daudztaktu procesors.

Kurss "Ievads digitālajā projektēšanā"

Lekcija 03.12.2010

Autors: **Artis Mednis**

Rīga 2010

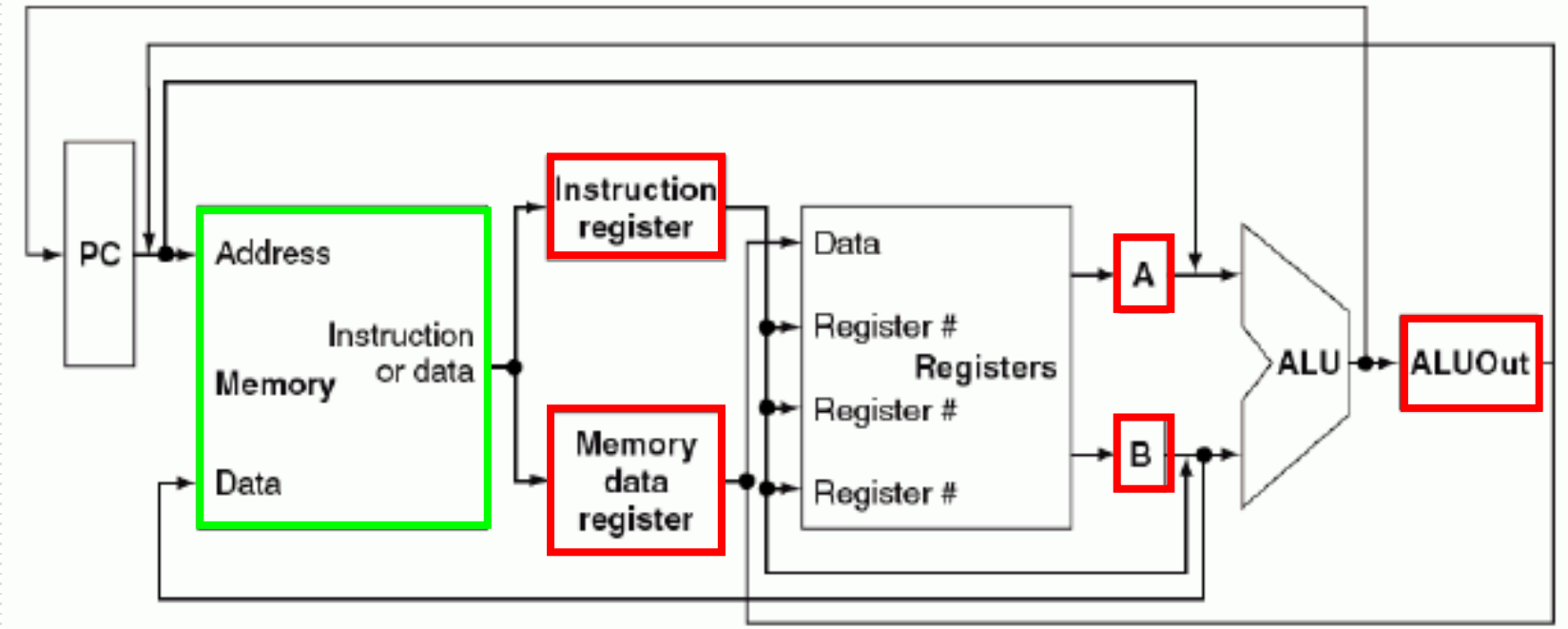
Daudztaktu risinājuma pamatprincipi

- Katrs vientakts risinājuma *solis* tagad tiek pie *savas* takts

- Atsevišķas komponentes vienas instrukcijas izpildes laikā tiek izmantotas atkārtoti:
  - Viena kopīga atmiņa tiek izmantota gan instrukcijām, gan datiem
  - Viens *ALU*, bez papildus summatoriem

- Aiz katra komponenta parādās viens vai vairāki reģistri, kur glabāt datus nākošajam takts ciklam

Daudztaktu procesors (vienkāršots)



Datu glabāšana

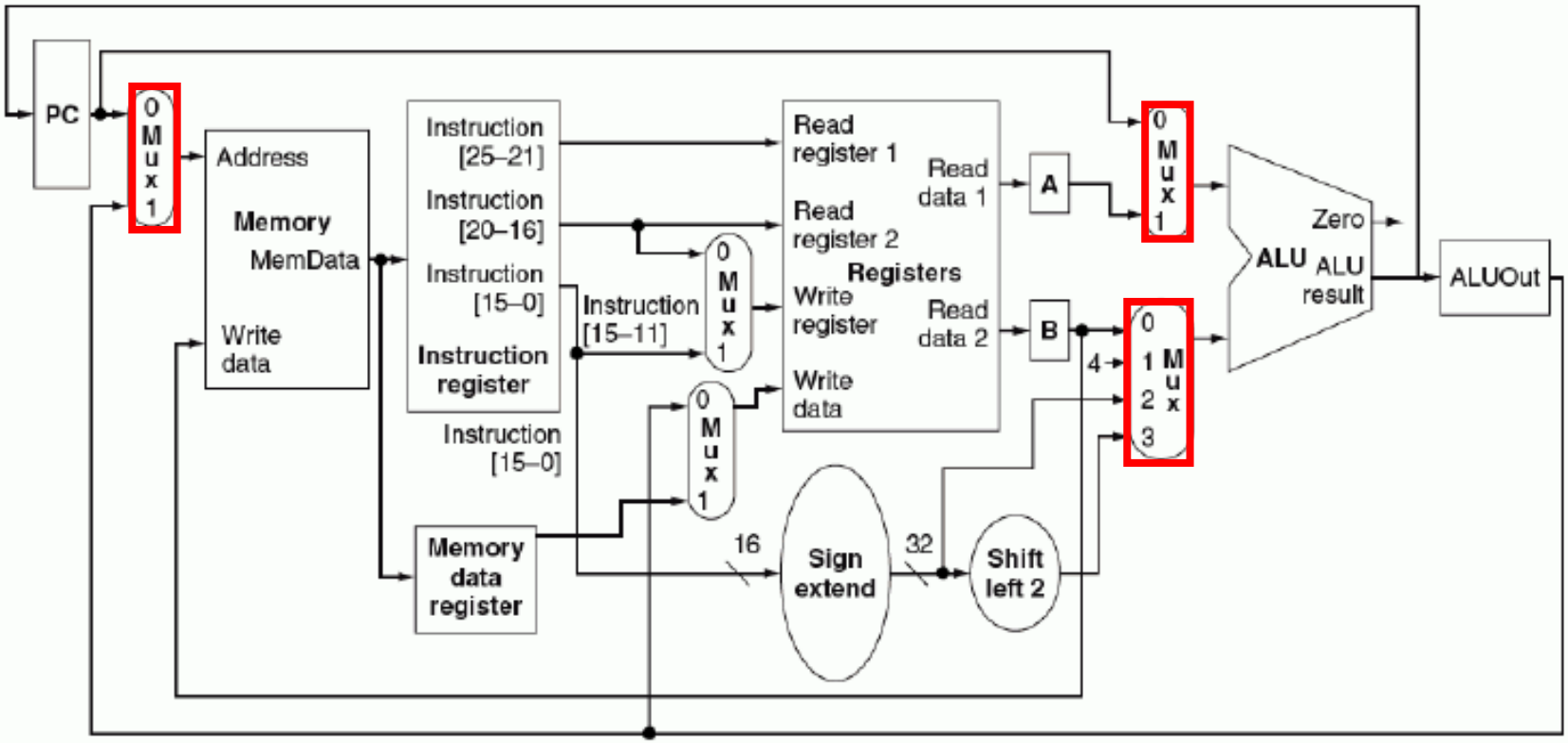
- Katras takts beigās ir nepieciešams saglabāt datus:
  - Nākošajām instrukcijām – reģistru failā, *PC* vai atmiņā
  - Šīs pašas instrukcijas nākošajām taktīm – papildus reģistros
- Kam tad vajag papildus reģistrus?
  - Atmiņai – instrukciju reģistrs un atmiņas datu reģistrs
  - Reģistru failam – *A* reģistrs un *B* reģistrs
  - *ALU* – *ALUOut* reģistrs
- Instrukciju reģistram vajag arī *Write Control* signālu – kādēļ tā? (diskusija)

Papildus multipleksori

- Atmiņas adresācijai:
  - Instrukcijām – no *PC*
  - Datiem – no *ALUOut*

- *ALU* operandiem:
  - *A* – no *A* reģistra vai *PC*
  - *B* – ieeju skaitu palielinām līdz 4:
    - *B* reģistrs
    - konstante 4 (*PC* aprēķinam)
    - paplašināts *offset* lauks (atmiņas adresēm)
    - paplašināts un pa kreisi nobīdīts *offset* lauks (*branch* adresēm)

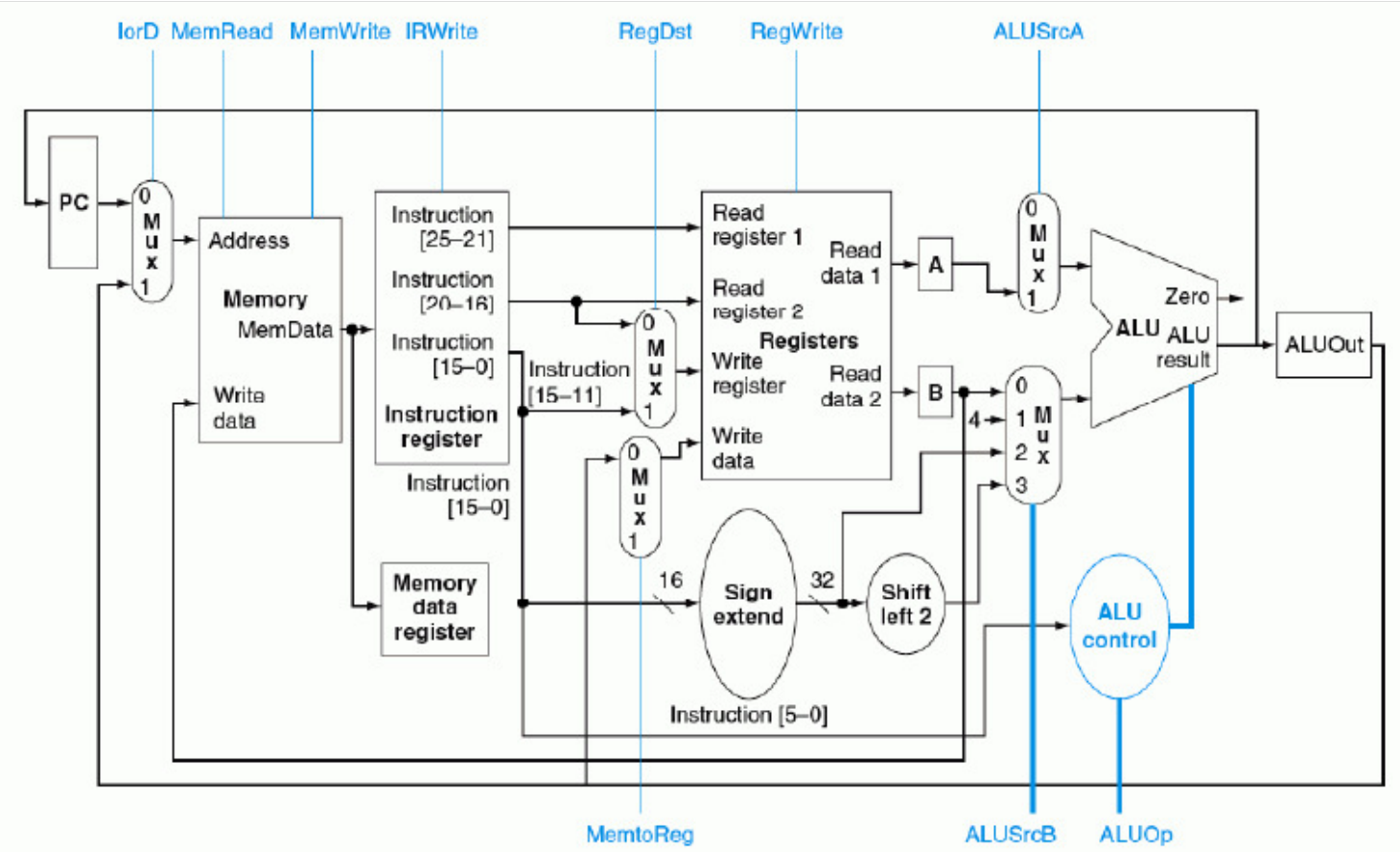
Daudztaktu procesors (jau detalizētāks)



Papildus kontroles signāli I

- *Write*
  - *PC*, atmiņa, reģistri
- *Read*
  - Atmiņa
- *ALU Control* – par laimi, derēs no vientakts procesora ☺
- Jaunie multipleksori – viena vai divas kontroles līnijas

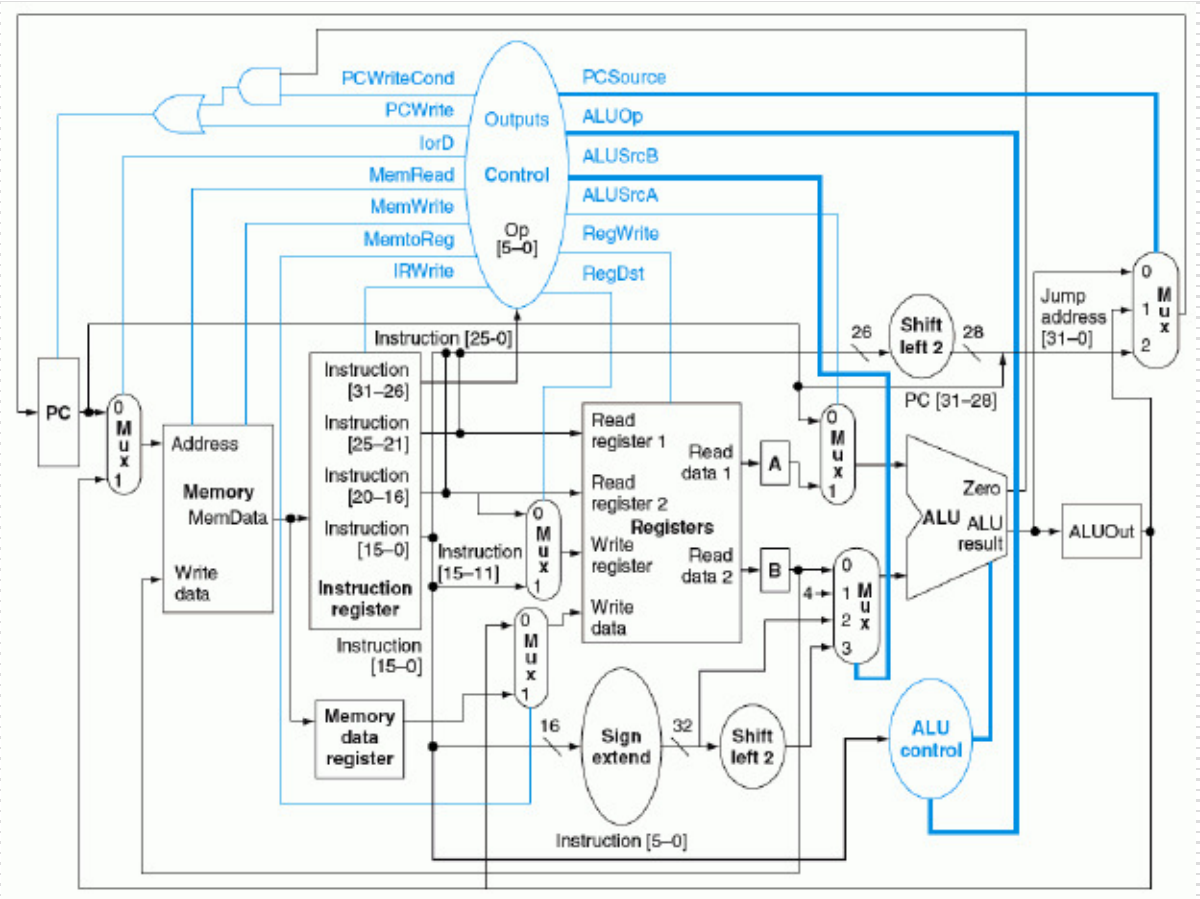
Papildus kontroles signāli II



Kas mums vēl pietrūkst?

- Nākošā *PC* vērtība:
  - *PC* + 4 no *ALU* (1)
  - *ALUOut* - *branch* gadījumā (2)
  - 26 biti no *IR* un 4 biti no *PC* + 4 *jump* gadījumā (3)
- PC vērtība var būt bez nosacījuma (1, 3) vai ar nosacījumu (2)
- Izmantojam divus atsevišķus signālus
  - *PCWrite*
  - *PCWriteCond*

Daudztaktu procesors (ar *branch* un *jump*)



- Pārtraukums 10 minūtes

Instrukciju sadale takts ciklos

- Pamatprincipi:
  - vienā ciklā viena *ALU* operācija
  - vai viena darbība ar reģistru failu
  - vai viena darbība ar atmiņu
- Pieci secīgi soļi:
  - *Instruction fetch*
  - *Instruction decode and register fetch*
  - *Execution, memory access or branch completion*
  - *Memory access or R-type instruction completion*
  - *Memory read completion*

Instruction fetch

- Nolasa instrukciju no atmiņas
- Izrēķina nākošās secīgās instrukcijas adresi

- Operācijas:
  - *PC* saturs tiek padots uz atmiņu kā adrese
  - Nolasītā instrukcija tiek ierakstīta *IR*
  - *PC* tiek palielināts par 4

Instruction decode and register fetch

- Šajā solī joprojām nav zināms, kas tad tā ir par instrukciju ☺ Bet tas nekas...
- Nolasa divus reģistrus atbilstoši laukiem *rs* un *rt* un saglabā datus reģistros *A* un *B*
- Izrēķina *branch* adresi un saglabā datus reģistrā *ALUOut*

Execution, memory access or branch completion

- Beidzot ir zināms, kas tā ir par instrukciju ☺
- *ALU* nostrādā, izmantojot datus no iepriekšējā soļa
- Variants #1 – tiek izrēķināta un saglabāta reģistrā *ALUOut* atmiņas adrese
- Variants #2 – tiek veikta aritmētiska vai loģiska operācija, datus saglabājot reģistrā *ALUOut*
- Variants #3 – tiek veikta salīdzināšana, lai izpildītu *branch* instrukciju, pozitīva lēmuma gadījumā uz *PC* tiek padoti dati no *ALUOut*
- Variants #4 – *PC* tiek ierakstīta *jump* adrese

Memory access or R-type instruction completion

- *Load* instrukcijas gadījumā dati no atmiņas tiek ierakstīti atmiņas datu reģistrā
- *Store* instrukcijas gadījumā dati no reģistra *B* tiek ierakstīti atmiņā
- Abos gadījumos tiek izmantota adrese, kura ir izrēķināta un saglabāta *ALUOut* iepriekšējos soļos

- R-tipa instrukcijas gadījumā dati no reģistra *ALUOut* tiek saglabāti instrukcijā norādītajā reģistrā *rd*

Memory read completion

- *Load* instrukcija tiek pabeigta, datus no atmiņas datu reģistra ierakstot instrukcijā norādītajā reģistrā *rd*

- Vēlreiz visi soļi vienā tabulā:

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			
Instruction decode/register fetch	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	if (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]],2'b00)}
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		
Memory read completion		Load: Reg[IR[20:16]] <= MDR		

Nākošajā lekcijā...

- Daudztaktu procesora kontroles modulis
- Galīgie automāti
- Stāvokļu pārejas diagrammas

Praktiskie darbi

- Strādājam pie kursa projekta KP3

Pateicos par uzmanību!

Jautājumi?