# Architektury systemów komputerowych

Lista zadań nr 13

Na zajęcia 4 i 5 czerwca 2018

**UWAGA!** W trakcie prezentacji zadań należy być przygotowanym do wytłumaczenia haseł, które zostały oznaczone **wytłuszczoną** czcionką.

W zadaniach odnoszących się do potokowej realizacji procesora MIPS zakładamy, że procesor nie implementuje *branch delay slots* chyba, że podano inaczej.

**Zadanie 1 (2 pkt.).** Na schemacie z następnej kartki widnieje jednocyklowa realizacja procesora MIPS. Przedstaw kodowanie instrukcji, stan sygnałów kontrolnych i modyfikacje niezbędne do obsługi dodatkowych instrukcji (a) «jr» i «lui», (b) «jal». Poniżej podano semantykę instrukcji:

| mnemonik | typ | semantyka |
|---|---|---|
| `jr $Rs` | R | `PC := Reg[Rs];` |
| `lui $Rt,imm` | I | `Reg[Rt] := imm << 16;` |
| `jal addr` | J | `Reg[31] := PC + 8;`<br>`PC := (PC & 0xf0000000) \| (addr << 2);` |

**Zadanie 2.** Przypuśćmy, że w jednocyklowej implementacji procesora MIPS jeden z sygnałów kontrolnych: (a) RegWrite, (b) $\texttt{ALUControl}_1$, albo (c) MemWrite; zostaje w wyniku błędu projektanta podpięty na stałe do napięcia wyrażającego logiczny stan 0 albo 1. Które z omówionych na wykładzie instrukcji przestaną działać i dlaczego? Rozważ obydwie wartości.

| ALUOp | Funct | $\texttt{ALUControl}_{2\ldots0}$ |
|---|---|---|
| 00 | xxxxxx | 010 (add) |
| x1 | xxxxxx | 110 (sub) |
| 1x | 100000 | 010 (add) |
| 1x | 100010 | 110 (sub) |
| 1x | 100100 | 000 (and) |
| 1x | 100101 | 001 (or) |
| 1x | 101010 | 111 (slt) |

| Instruction | Opcode | RegWrite | RegDst | ALUSrc | Branch | MemWrite | MemtoReg | ALUOp |
|---|---|---|---|---|---|---|---|---|
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 00 |
| sw | 101011 | 0 | x | 1 | 0 | 1 | x | 00 |
| beq | 000100 | 0 | x | 0 | 1 | 0 | x | 01 |

**Zadanie 3.** Procesory ARM należące do rodziny procesorów **RISC** oferują **tryb adresowania** z postinkrementacją. Spróbujmy dodać instrukcję «`lwai $rt, ($rs), imm`» o semantyce:

```
Reg[Rt] := Mem[Reg[Rs]];
Reg[Rs] := Reg[Rs] + imm << 16;
```

... do jednocyklowej implementacji procesora MIPS. Przedstaw propozycję kodowania, stan sygnałów kontrolnych i modyfikacje niezbędne do obsługi tej instrukcji. Jakie problemy sprawia dodanie tego rozkazu?

**Zadanie 4.** Na potokowej implementacji procesora MIPS wykonujemy poniższe dwa programy:

```
1  lw   $1,40($2)
2  add  $2,$3,$3
3  add  $1,$1,$2
4  sw   $1,20($2)
```

```
1  add  $1,$2,$3
2  sw   $2,0($1)
3  lw   $1,4($2)
4  add  $2,$2,$1
```

- Znajdź wszystkie **zależności danych** typu **Read–After–Write**[1].
- Znajdź wszystkie **hazardy danych**, które wystąpią w implementacji potoku bez i z **obejściami**.
- Narysuj diagram stanu potoku (jak na slajdzie 61) dla wykonania powyższych ciągów instrukcji. Oznacz **wstrzymania potoku** oraz obejścia, z których korzysta przetwarzanie rozkazów.

**Zadanie 5.** Poniższe ciągi instrukcji wykonujemy na potokowej implementacji procesora MIPS.

```
1  lw   $1,40($6)
2  add  $2,$3,$1
3  add  $1,$6,$4
4  sw   $2,20($4)
5  and  $1,$1,$4
```

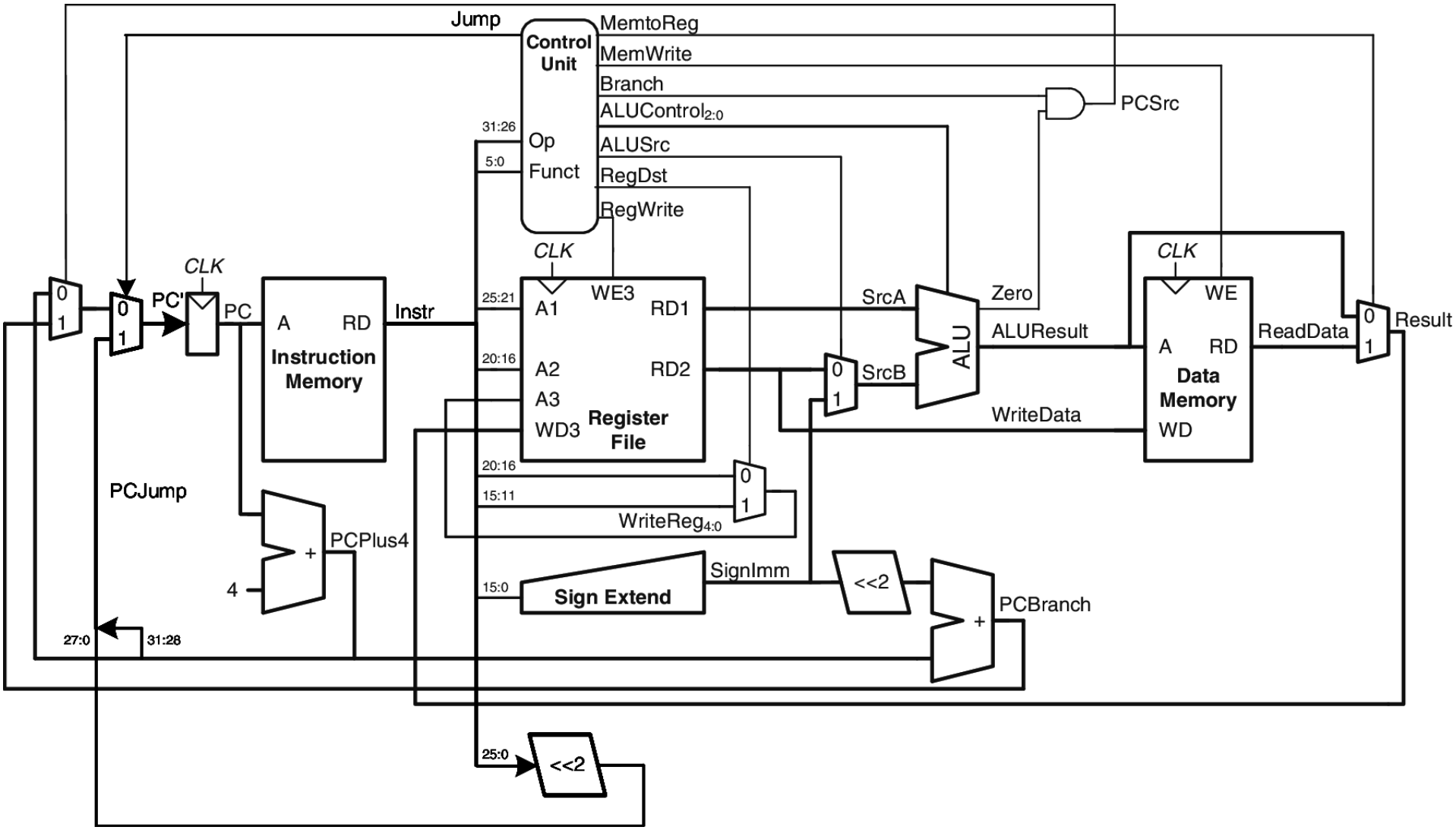
```
1  add  $1,$5,$3
2  sw   $1,0($2)
3  lw   $1,4($2)
4  add  $5,$5,$1
5  sw   $1,0($2)
```

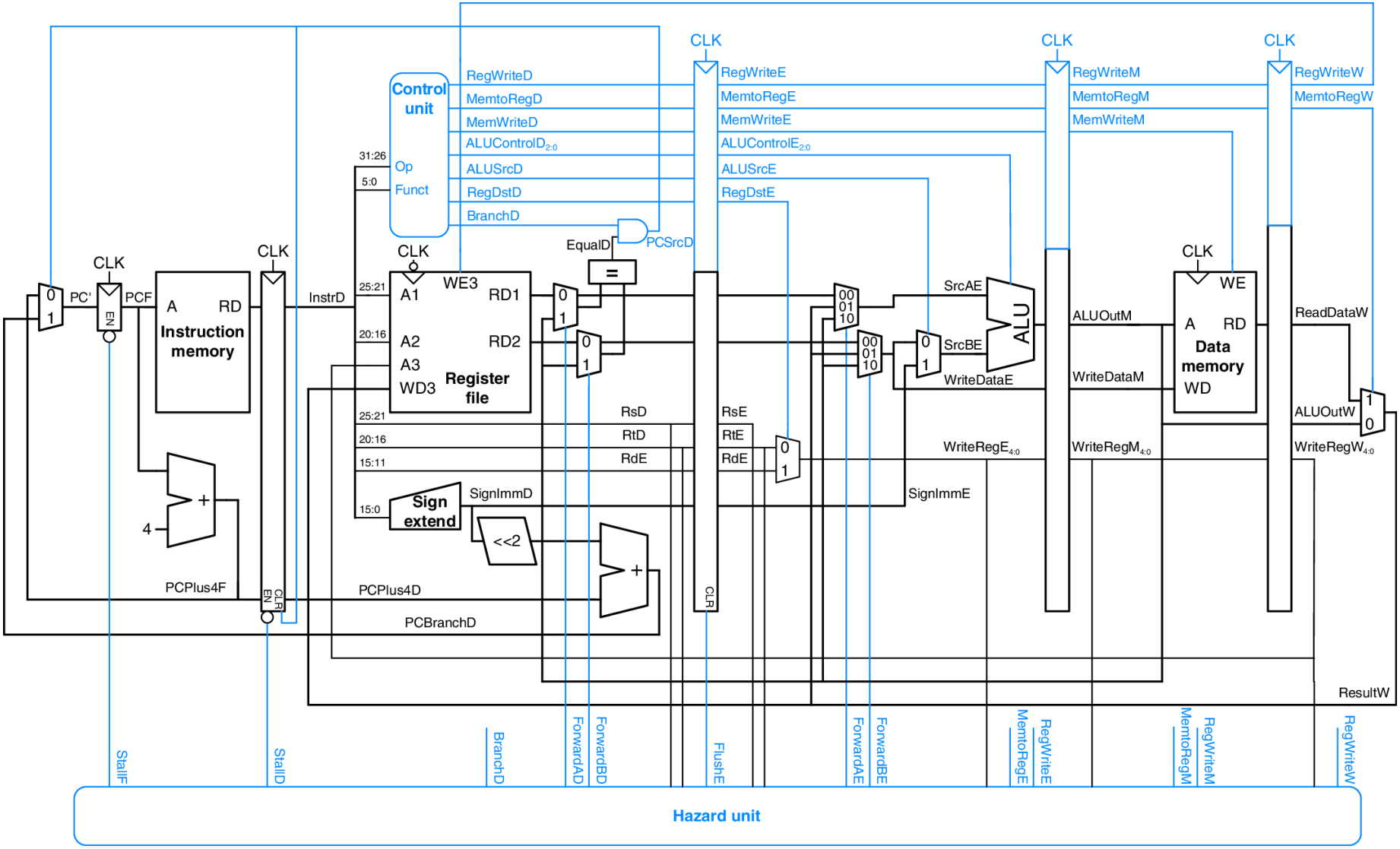
- Procesor nie ma zaimplementowanych objeść i **wykrywania hazardów**. Wstaw minimalną ilość rozkazów `nop`, aby zapewnić poprawność wykonania powyższych ciągów instrukcji.
- Powtórz poprzednie polecenie, ale postaraj się usunąć instrukcje `nop`. Zachowując semantykę możesz dokonywać dowolnych zmian w kodzie (dodawanie, usuwanie, zmiana kolejności instrukcji). Dodatkowo możesz używać rejestru $7 do przechowywania wartości tymczasowych.
- Procesor implementuje obejścia, ale nie ma wykrywania hazardów. W jaki sposób popsuje się semantyka powyższego kodu?

**Zadanie 6.** Na następnej kartce podano schemat potokowej realizacji procesora MIPS, do której chcemy dodać obsługę rozkazu «`slti`». Semantykę i kodowanie tej instrukcji podano w dokumencie „MIPS Reference Data Card" dostępnym na stronie wykładu. Opisz stan sygnałów kontrolnych i modyfikacje niezbędne do obsłużenia rozkazu «`slti`». Jak należy zmodyfikować jednostkę wykrywania i zapobiegania hazardom?

**Zadanie 7.** Powtórz poprzednie zadanie dla instrukcji skoku `j`. Zwróć szczególną uwagę na akcję **przeładowania potoku** podejmowaną przez procesor w trakcie wykonywania skoku w fazie EX.

[1] W procesorach superskalarnych będą jeszcze hazardy danych *Write–After–Read* i *Write–After–Write*.


Jump
Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
PCSrc
31:26
5:0
Op
Funct
CLK
CLK
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
A
RD
Data Memory
WD
ReadData
Result
PCJump
20:16
15:11
WriteReg4:0
PCPlus4
4
+
15:0
Sign Extend
SignImm
<<2
PCBranch
+
27:0
31:28
25:0
<<2


CLK
Control unit
Op
Funct
31:26
5:0
RegWriteD
MemtoRegD
MemWriteD
ALUControlD2:0
ALUSrcD
RegDstD
BranchD
EqualD
PCSrcD
=
RegWriteE
MemtoRegE
MemWriteE
ALUControlE2:0
ALUSrcE
RegDstE
RegWriteM
MemtoRegM
MemWriteM
RegWriteW
MemtoRegW
PC'
PCF
EN
A
RD
Instruction memory
InstrD
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register file
RsD
RtD
RdE
RsE
RtE
RdE
15:11
15:0
Sign extend
SignImmD
SignImmE
<<2
4
+
PCPlus4F
PCPlus4D
PCBranchD
CLR
SrcAE
SrcBE
WriteDataE
ALU
ALUOutM
WriteDataM
WE
WD
Data memory
ReadDataW
ALUOutW
WriteRegE4:0
WriteRegM4:0
WriteRegW4:0
ResultW
StallF
StallD
BranchD
ForwardAD
ForwardBD
FlushE
ForwardAE
ForwardBE
MemtoRegE
RegWriteE
MemtoRegM
RegWriteM
RegWriteW
Hazard unit