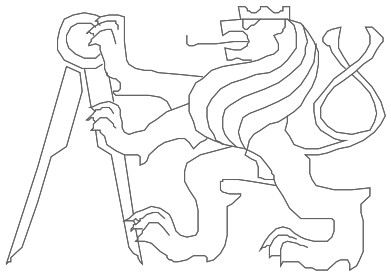
Architektura počítačů

Procesor

V přednášce byly použity (se souhlasem vydavatelství) obrázky z knihy Paterson, D., Henessy, V.: Computer Organization and Design, The HW/SW Interface. Elsevier, ISBN: 978-0-12-370606-5



České vysoké učení technické, Fakulta elektrotechnická

Počítač podle von Neumanna tvoří

- Řadič
- ALU
- Paměť
- Vstup
- Výstup

Procesor/mikroprocesor

Harvardská architektura je variantou s oddělenou pamětí programu a pamětí dat

V/V podsystém

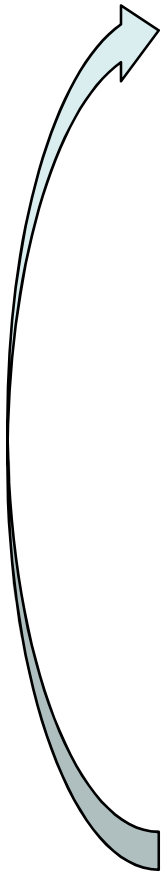
Řadič - součást (jednotka) počítače/procesoru, která jeho činnost řídí.
Sestává ze dvou částí:

- datové
  - registry,
  - další potřebné obvody,
- vlastní řídicí části, z tzv. jádra řadiče.

Důležité registry řadiče

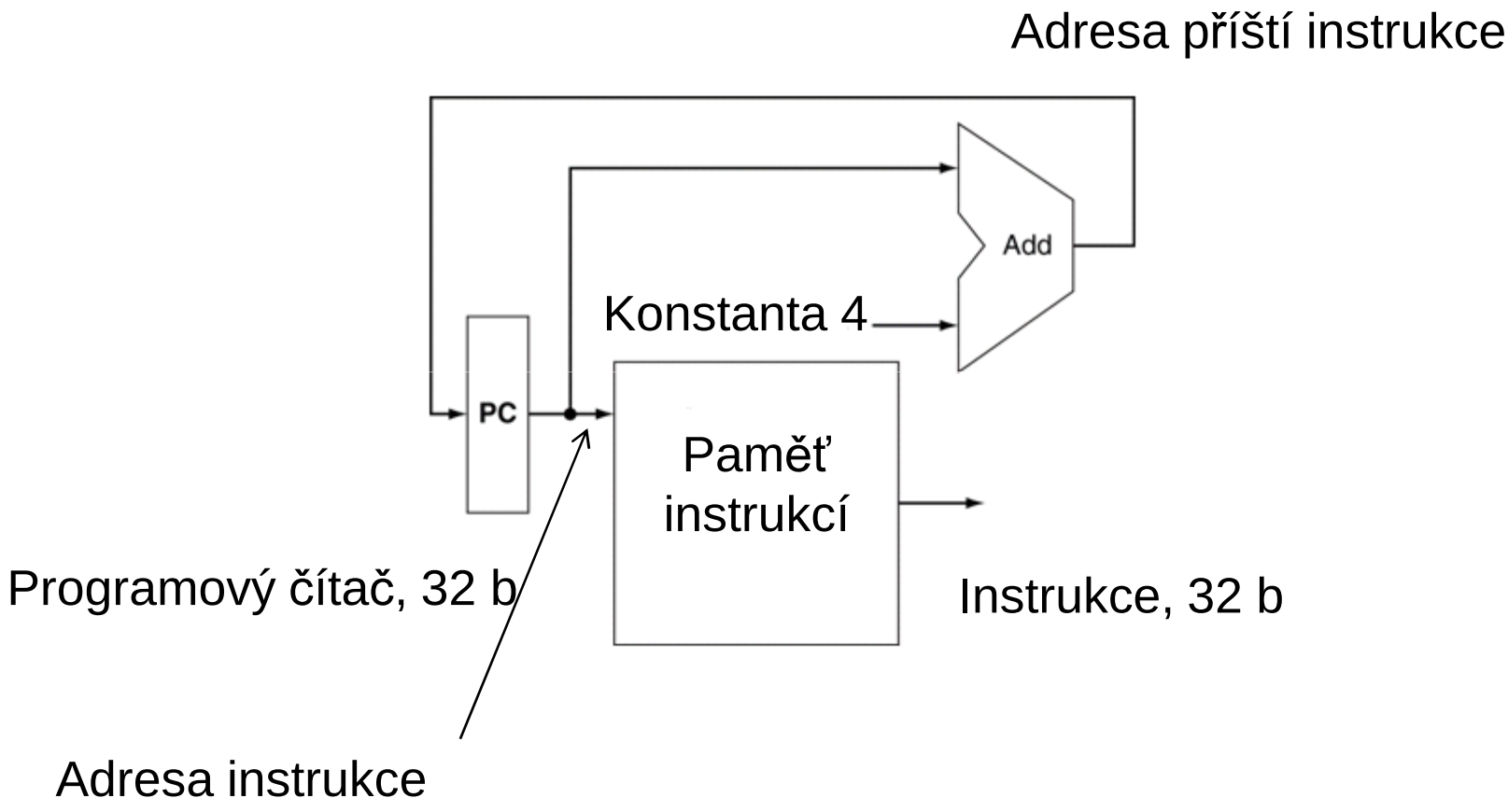
- PC (Program Counter), programový čítač.
- IR (Instruction Register), registr instrukce
- Další
  - Univerzální nebo pracovní registry,
  - SP (Stack Pointer), ukazatel zásobníku,
  - PSW Program Status Word), stavové slovo programu,
  - IM (Interrupt Mask), maska přerušení.

Základní cyklus počítače – sekvenční postup vykonávání instrukcí



1. Počáteční nastavení, zejména např. PC.
2. Čtení instrukce
   - PC → adresa HP,
   - Čtení obsahu,
   - Přečtená data → IR,
   - PC+*l* → PC, kde *l* je délka instrukce.
3. Dekódování operačního znaku (OZ),
4. provedení operace (včetně vyhodnocení efektivních adres, čtení operandů, apod.).
5. Dotaz na možné přerušení. Ano-li, obsluha.
6. Ne-li, opakování od bodu 2.

Obvodová realizace základního cyklu počítače



Úkol pro tuto přednášku:

- Porozumět implementaci jednoduchého počítače tvořeného procesorem, oddělenými pamětmi instrukcí a dat a ALU, který umí instrukce
  - Čtení a zápis do datové paměti `lw` a `sw`,
  - Aritmetické-logické instrukce `add`, `sub`, `and`, `or` a `slt` a
  - Skokové instrukce `beq`.
- V procesoru bude řídicí jednotka (řadič) i ALU.
- Poznámka:
  - Na této přednášce jej budeme implementovat jednoduše (jako jednocyklový),
  - Na 4. přednášce ukážeme více realistickou, zřetězenou verzi.

Formát instrukcí

- Uvažujme tři typy instrukcí dle tabulky:

Typ	31…					0
R	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**rd**(5), 15:11	**shamt**(5)	**funct**(6), 5:0
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0		
J	**opcode**(6), 31:26	**address**(26), 25:0				

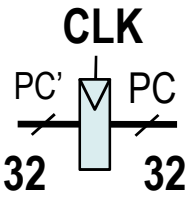
- všechny R instrukce -> opcode=000000, funct – operace
- rs – source, rd – destination, rt – source/destination
- shamt – při operacích posunu, immediate – přímý operand

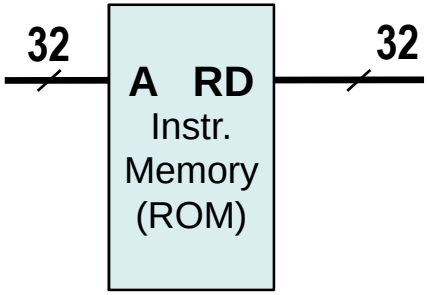
- K dispozici je 32 pracovních registrů

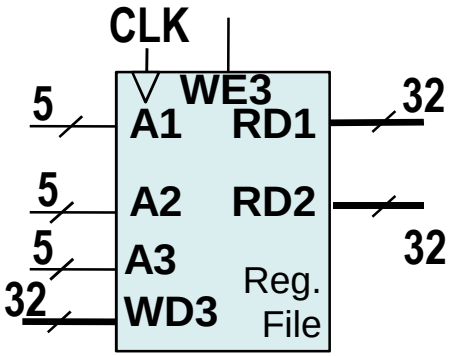
Kódování OPcode

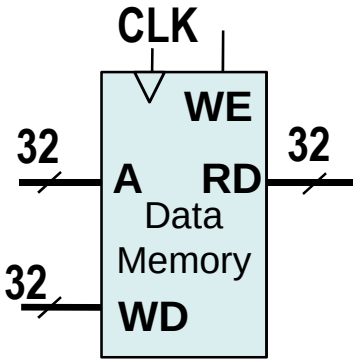
ALUOp	Funct	ALUControl
00	X	010 (add)
01	X	110 (sub)
1X	add (100000)	010 (add)
1X	sub (100010)	110 (sub)
1X	and (100100)	000 (and)
1X	or (100101)	001 (or)
1X	slt (101010)	111 (set less than)

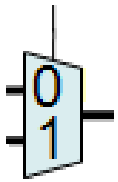
K dispozici máme tyto stavební prvky







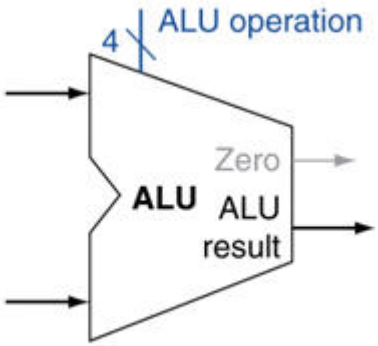


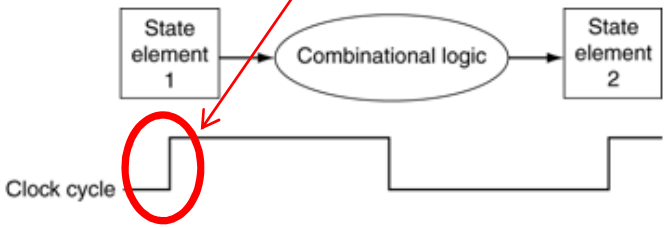


Multiplexor

Zápis náběžnou hranou CLK při WE = 1

Čtení po uplynutí „dostatečně dlouhé“ doby





Výklad syntaxe a sémantiky instrukce: například `lw`

`lw` – load word - čtení slova z datové paměti

Description	A word is loaded into a register from the specified address
Operation:	$t = MEM[$s + offset];
Syntax:	lw $t, offset($s)
Encoding:	`1000 11ss ssst tttt iiii iiii iiii iiii`

Uložme slovo z paměti na adrese 0x4 do registru č.11:

lw $11, 0x4($0)

```
1000 11ss ssst tttt iiii iiii iiii iiii
1000 1100 0000 1011 0000 0000 0000 0100
```

0 11 4

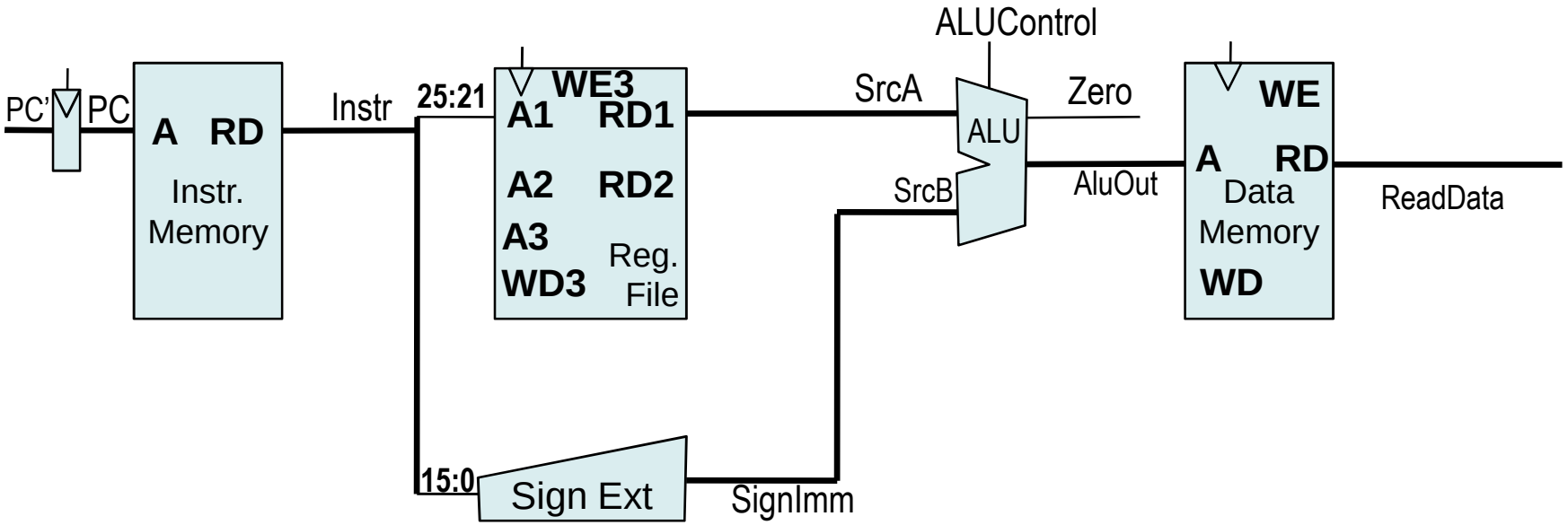
0x 8C 0B 00 04 – strojový kód instrukce lw $11, 0x4($0)

Poznámka: V registu $0 je trvale uložena konstanta 0.

Jedno-cyklový procesor – návrh – podpora čtení z paměti

- **`lw`**: typ I, rs – bázová adresa, imm – offset, rt – kde uložit

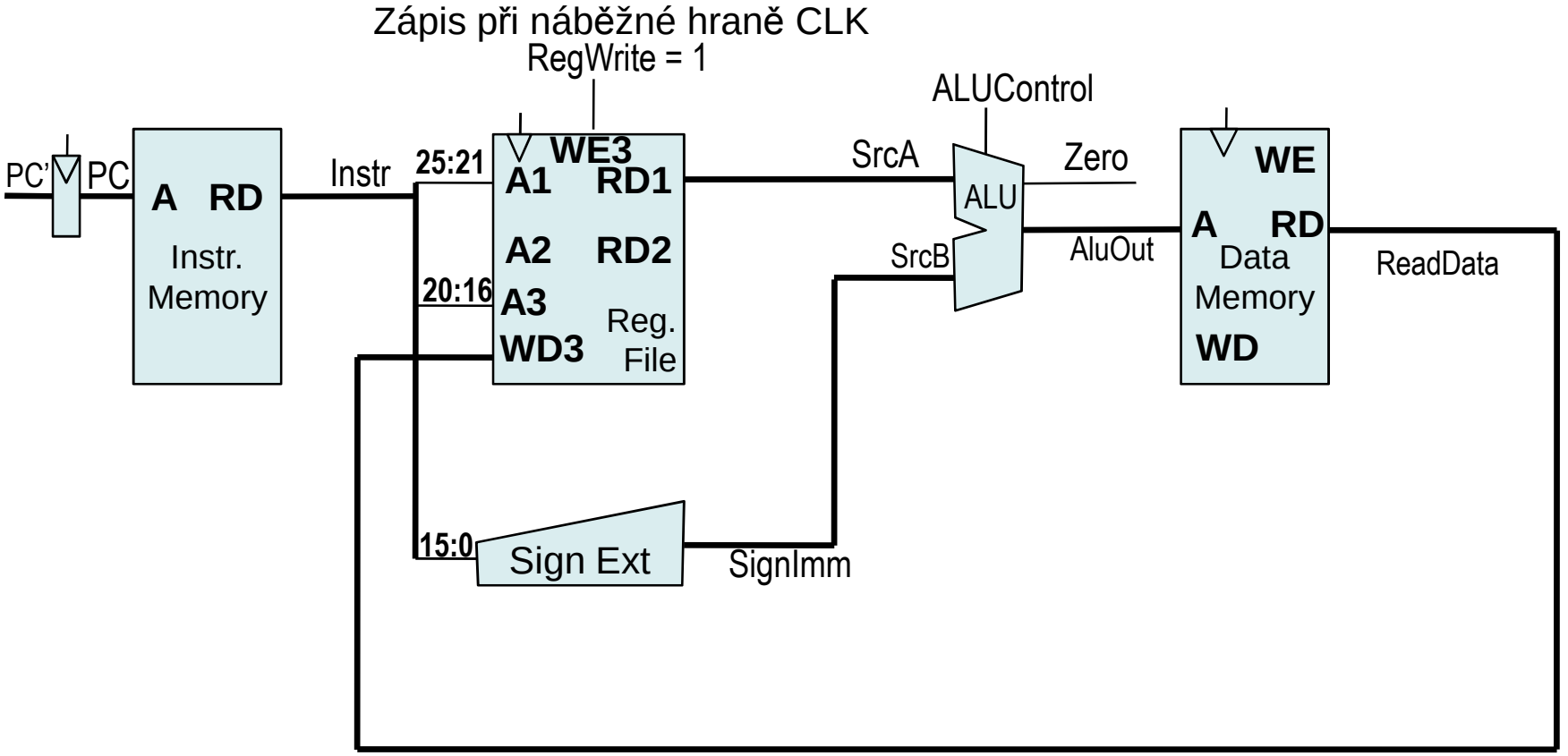
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora čtení z paměti

- **`lw`**: typ I, rs – bázová adresa, imm – offset, rt – kde uložit

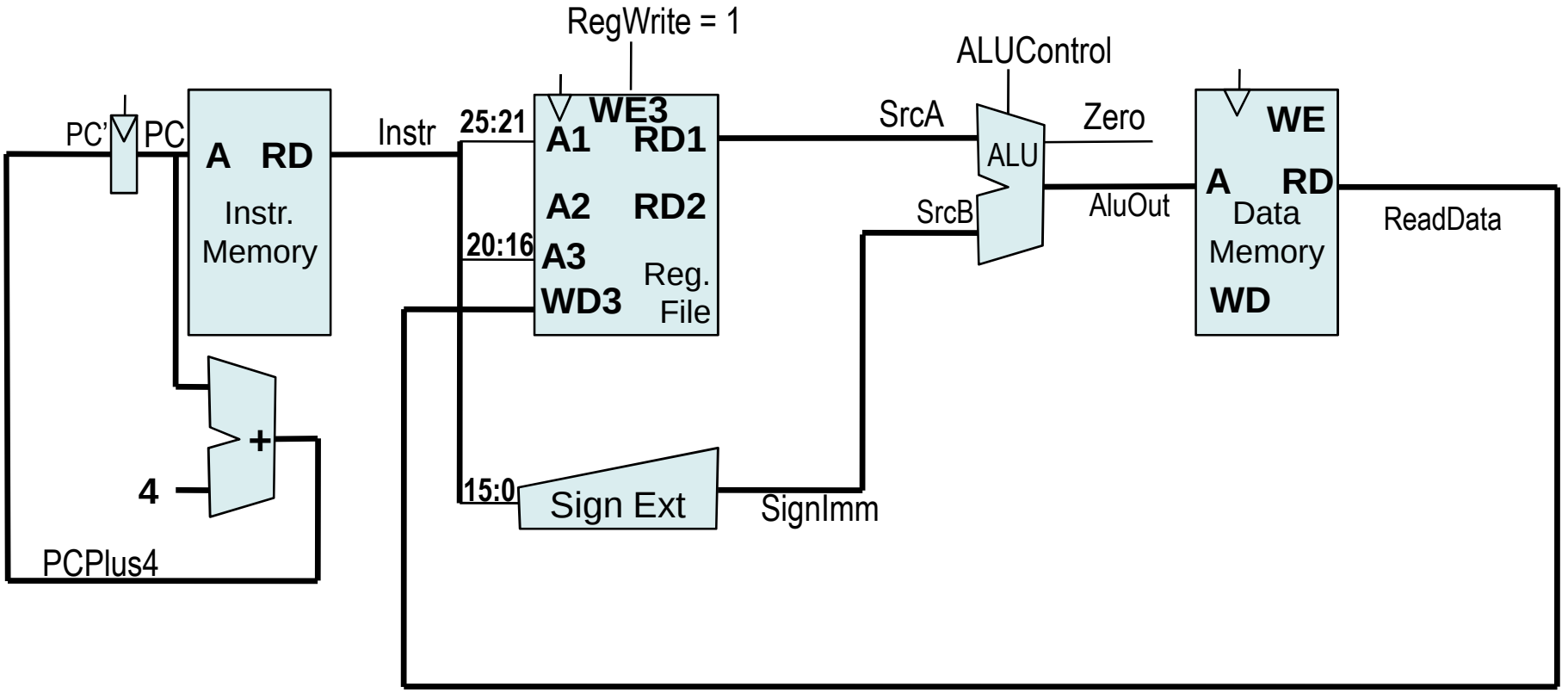
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora čtení z paměti

- **`lw`**: typ I, rs – bázová adresa, imm – offset, rt – kde uložit

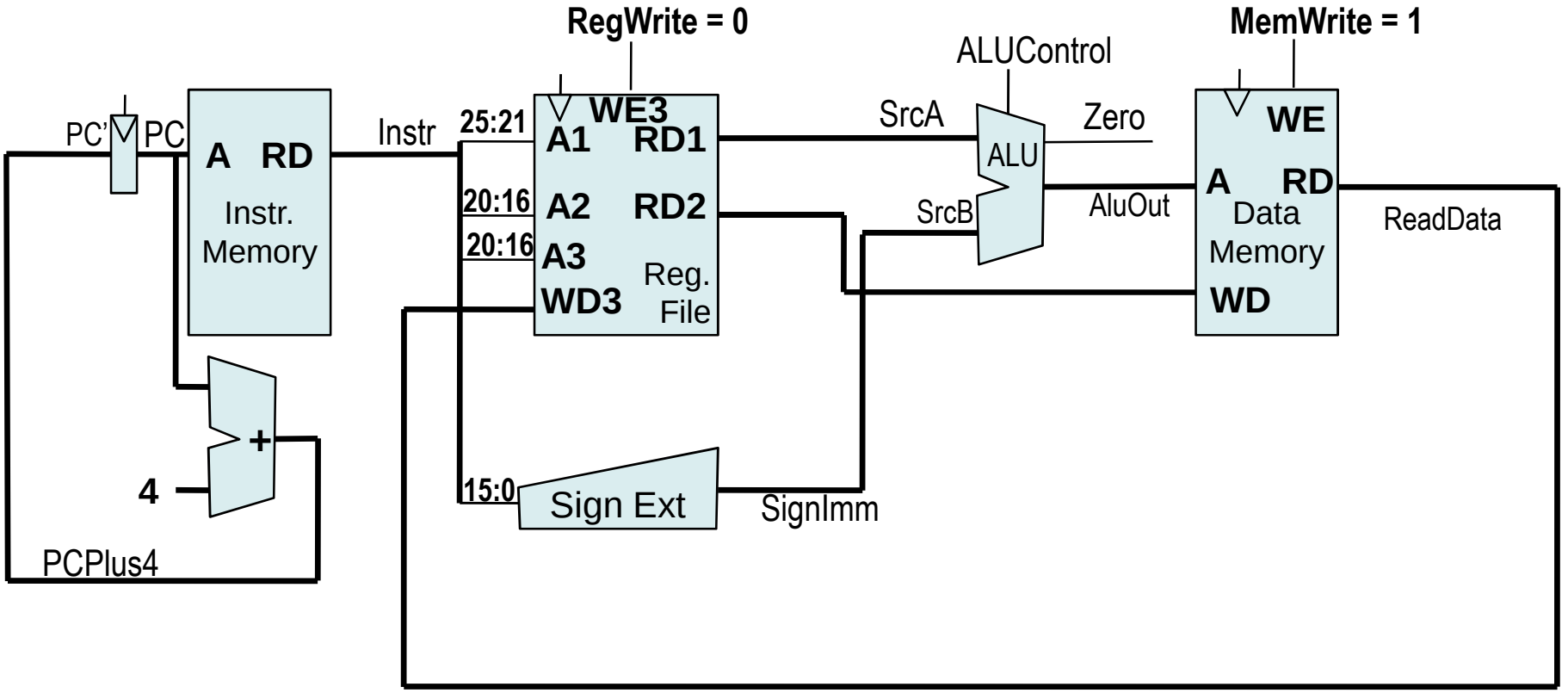
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora zápis do paměti

- **sw**: typ I, rs – bázová adresa, imm – offset, rt – co zapsat

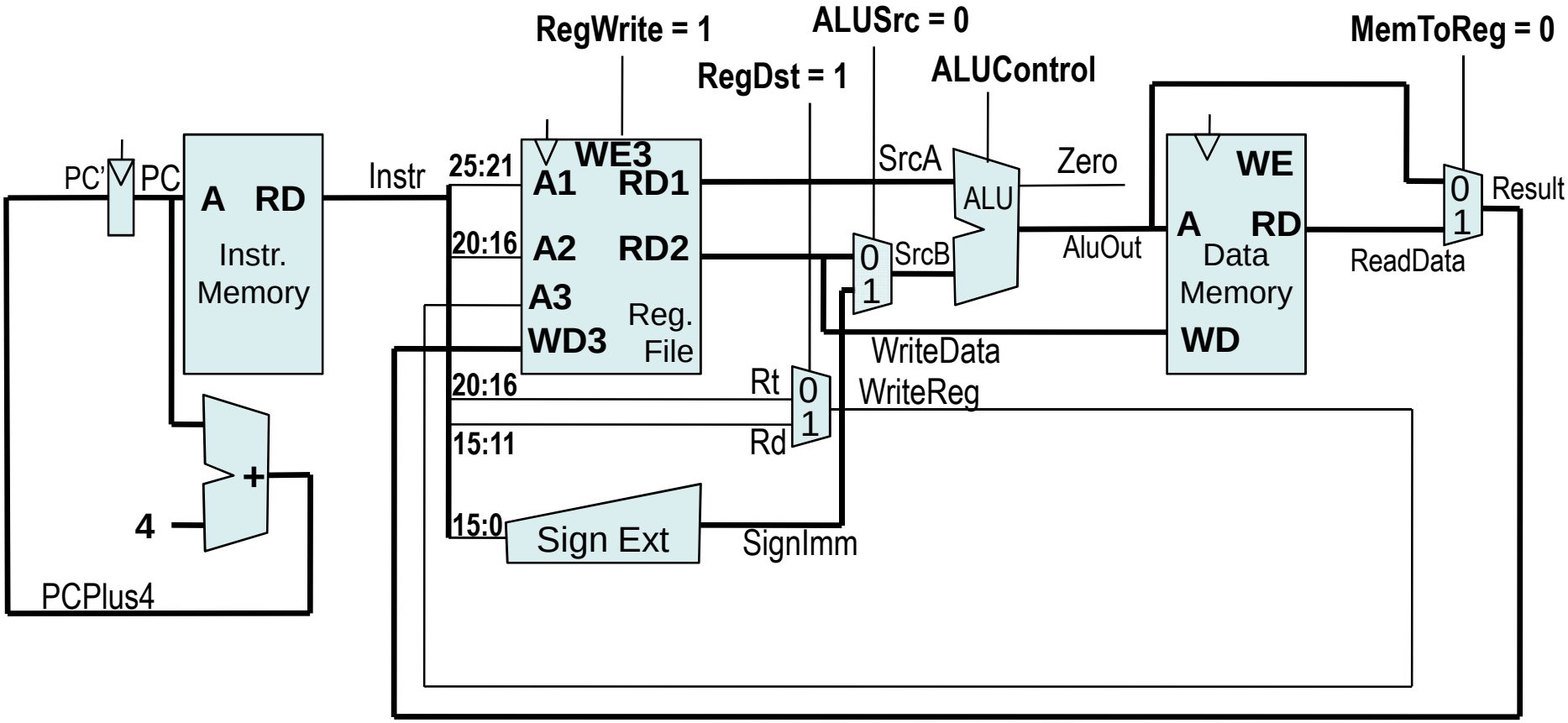
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora add

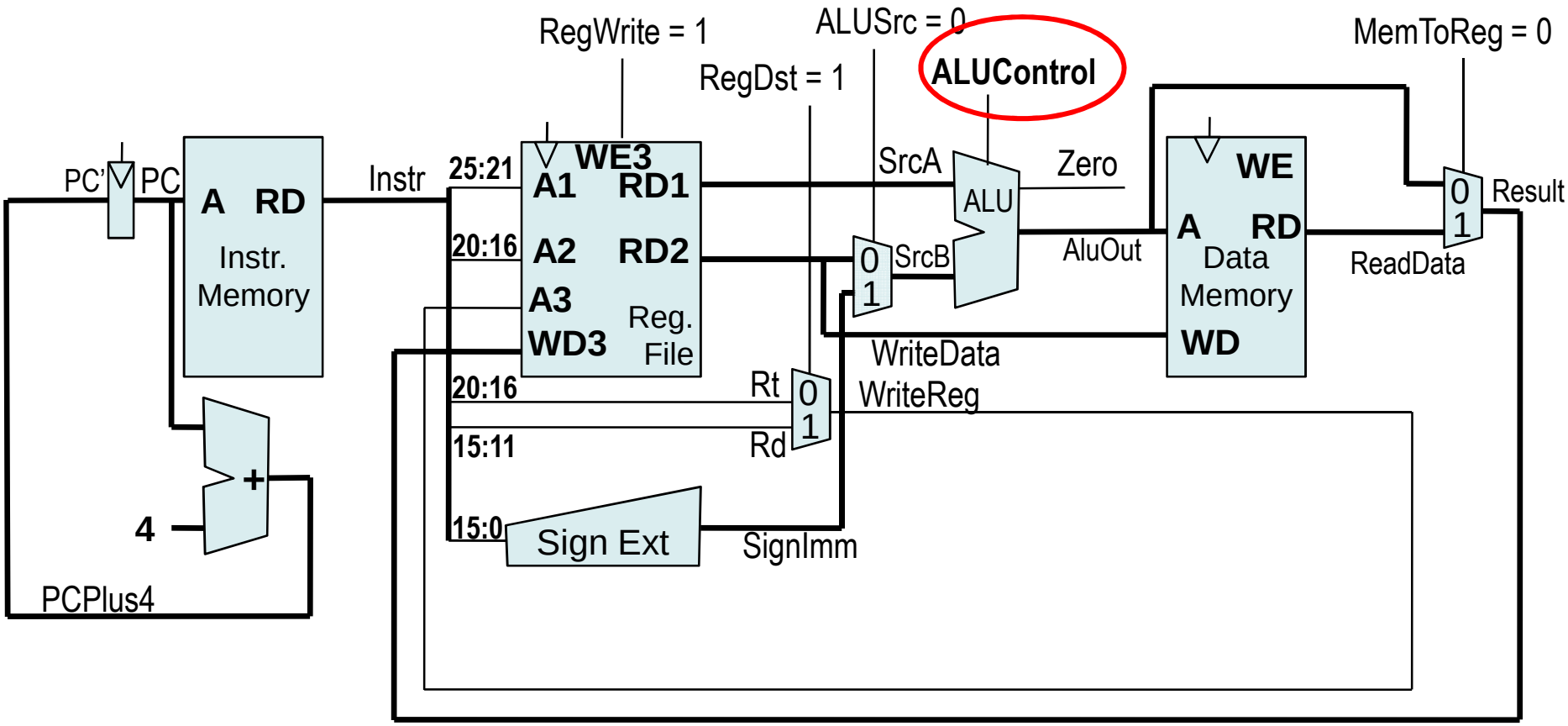
- **add**: typ R; rs, rt – zdroje, rd – cíl, funct – operace součtu

R	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**rd**(5), 15:11	**shamt**(5)	**funct**(6), 5:0



Jedno-cyklový procesor – návrh – podpora `sub`, `and`, `or`, `slt`

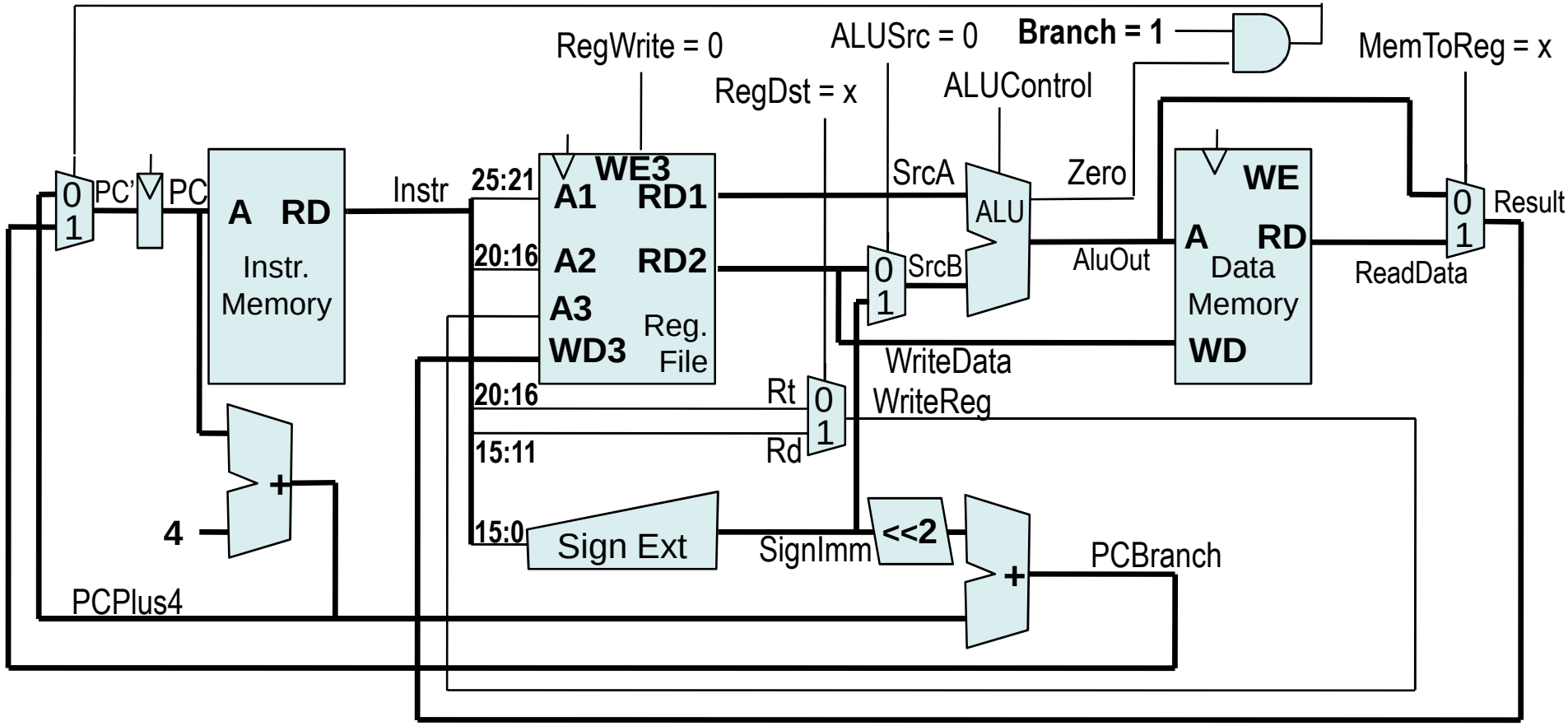
- jediné v čem se liší od `add` je operace ALU -> datapath beze změny; rozdíl v ALUControl



Jedno-cyklový procesor – návrh – podpora beq

- **beq** – branch if equal; imm–offset; PC´ = PC+4 + SignImm*4

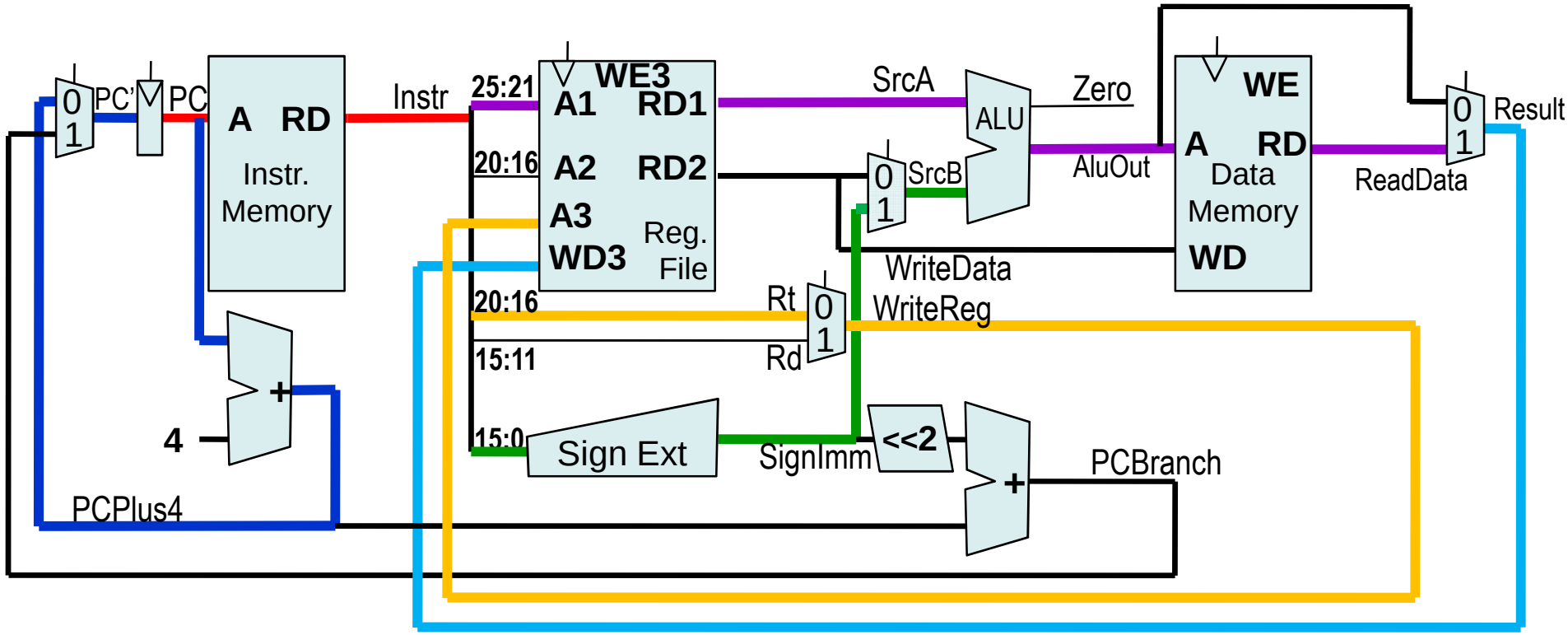
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – výkon: IPS = IC / T = $IPC_{str}.f_{CLK}$

- Jaká může být maximální frekvence procesoru?
- Zpoždění na kritické cestě – instrukce `lw`:

$$Tc = t_{PC} + t_{Mem} + t_{RFread} + t_{ALU} + t_{Mem} + t_{Mux} + t_{RFsetup}$$



Jedno-cyklový procesor – výkon: IPS = IC / T = $IPC_{str}.f_{CLK}$

- $Tc = t_{PC} + t_{Mem} + t_{RFread} + t_{ALU} + t_{Mem} + t_{Mux} + t_{RFsetup}$
- Předpokládejme:

  t_{PC} = 30 ns
  t_{Mem} = 300 ns
  t_{RFread} = 150 ns
  t_{ALU} = 200 ns
  t_{Mux} = 20 ns
  $t_{RFsetup}$ = 20 ns

  Pak Tc = 1020 ns --> $f_{CLK\ max}$ = 980 kHz,

  IPS = 1. 980e3 = 980 000 instrukcí za sekundu

Důležitá poznámka

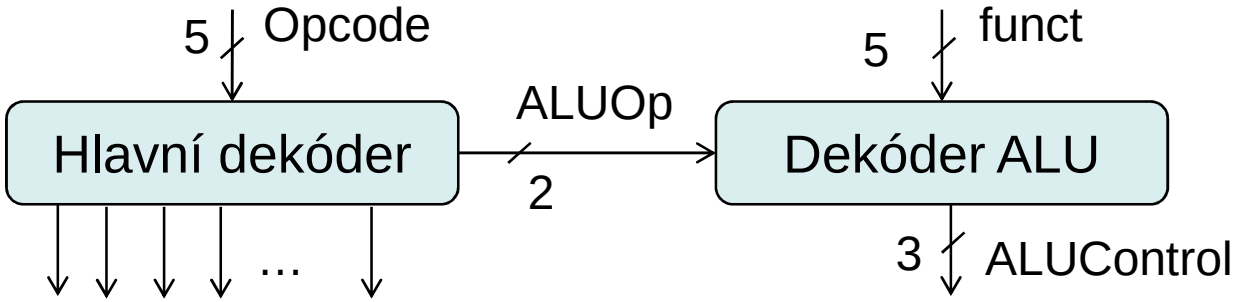
- Tenhle výsledek si, prosím, zapamatujte.
- Budeme s ním pracovat na 4. přednášce.

- Dnes se dále budeme zabývat porozuměním funkci **řadiče**.

Jedno-cyklový procesor – návrh – řídicí část

R	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**rd**(5), 15:11	**shamt**(5)	**funct**(6), 5:0
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0		
J	**opcode**(6), 31:26	**address**(26), 25:0				

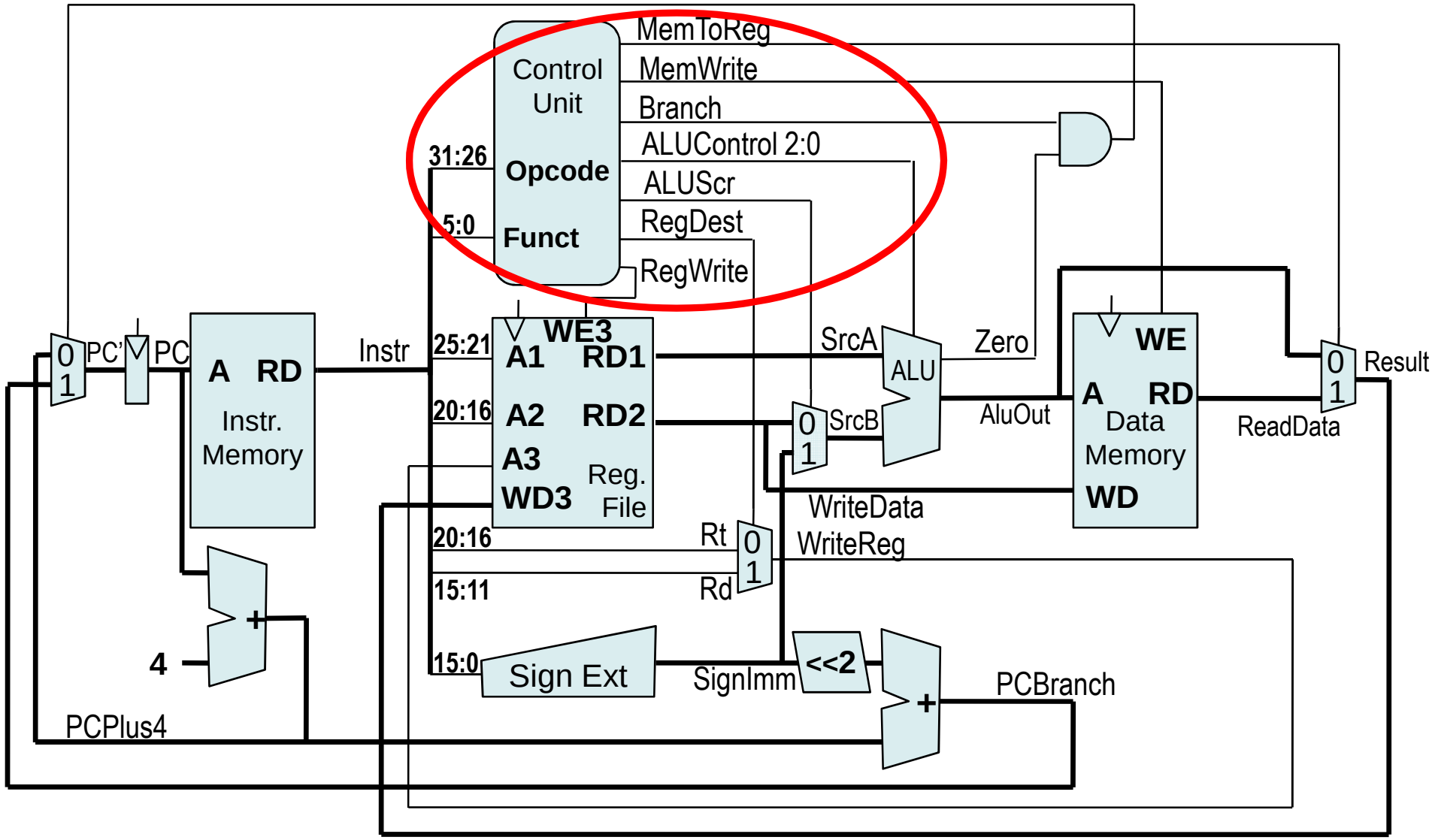
- řídicí signály na základe **opcode** a **funct**



ALUOp	
00	součet
01	rozdíl
10	podle funct
11	-nepoužito-

	Opcode	RegWrite	RegDst	ALUSrc	ALUOp	Branch	Mem Write	MemTo Reg
R typ	000000	1	1	0	10	0	0	0
lw	100011	1	0	1	00	0	0	1
sw	101011	0	X	1	00	0	1	X
beq	000100	0	X	0	01	1	0	X

Řadič jedno-cyklového procesoru



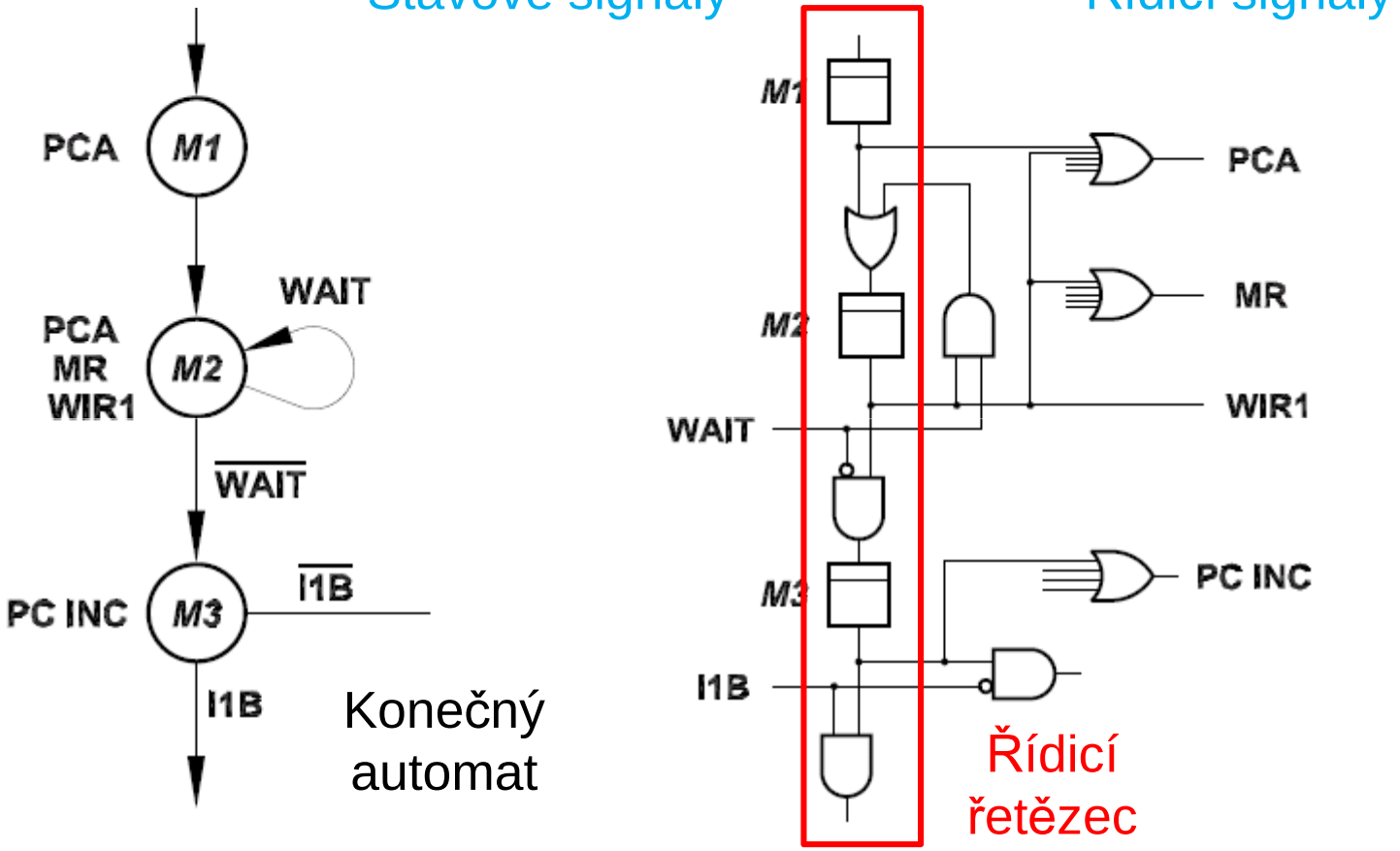
Co je řadič procesoru?

- Funkce řadiče: V příslušný časový okamžik generovat řídicí signály a přijímat signály stavové.
- Řadič — jednotka/sekvenční obvod,
  - výstupy: řídicí signály,
  - vstupy: stavové signály.
- Poznámka pro náš specifický případ: náš řadič reaguje např. na stavový signál Zero.

Možné realizace řadiče

- Řadič klasický, též obvodově realizovaný, tedy tzv. „obvodový“:
  - řadič s řídicími řetězci,
  - řadič na bázi čítače,
  - jinak navržený.
- Řadič mikroprogramovaný (řízený mikroprogramem):
  - horizontální,
  - vertikální,
  - diagonální.

Realizace: řadič s řídicími řetězci

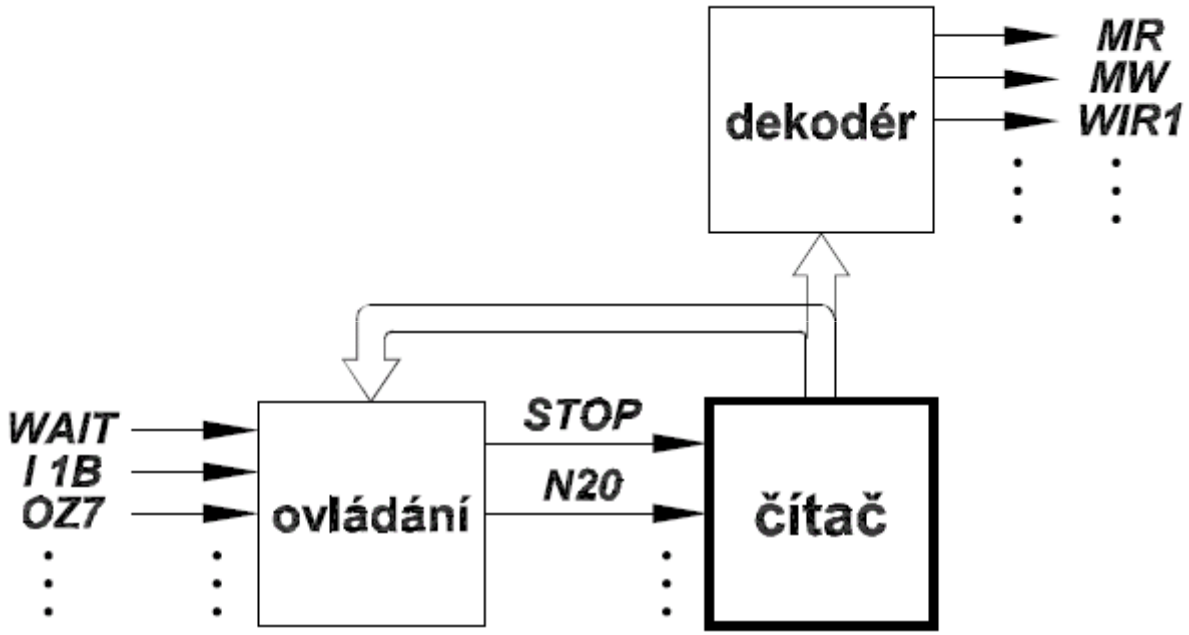


Důležitá poznámka: označení stavů a názvy řídicích a stavových signálů na obrázku **neodpovídají** našemu specifickému případu!

Řadič na bázi čítače

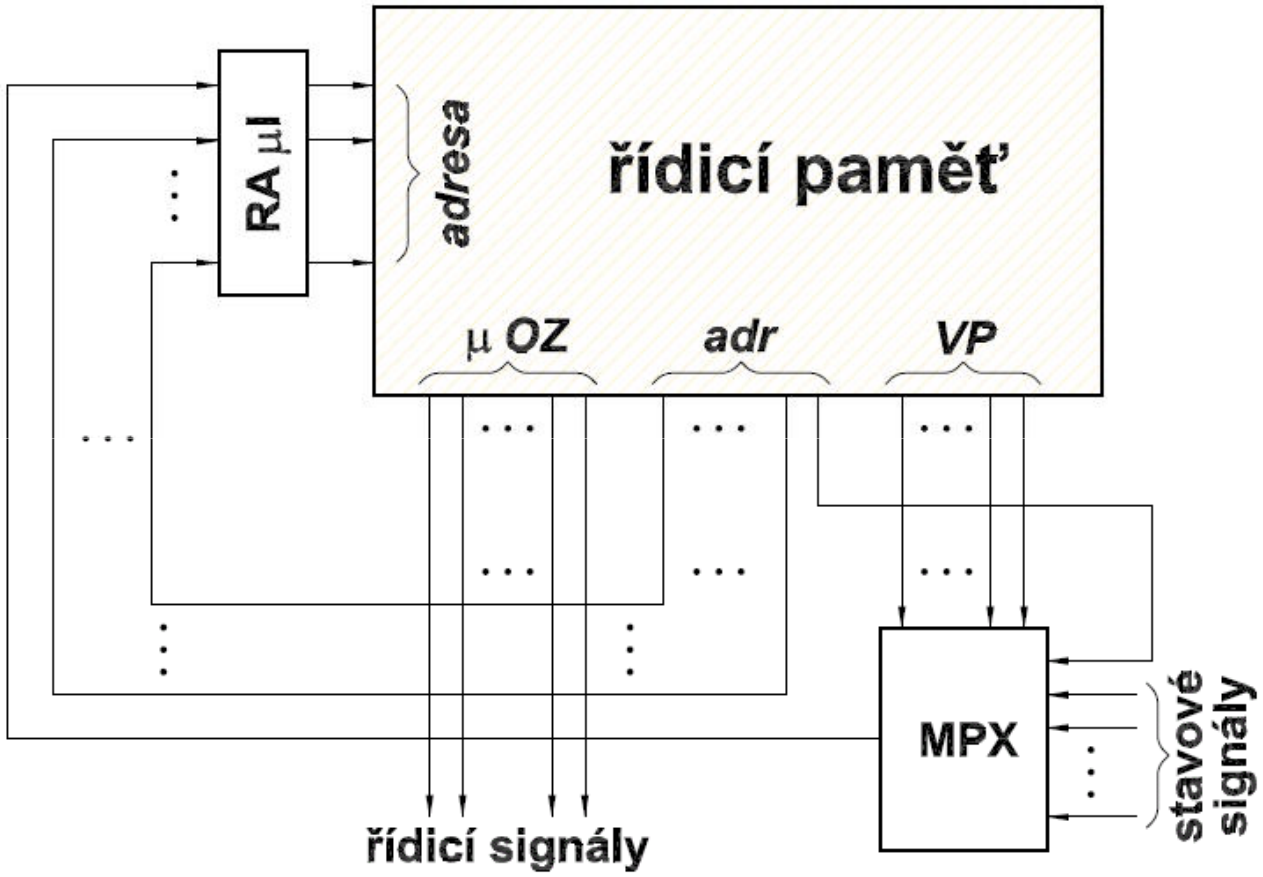
Stavové signály

Řídicí signály



Důležitá poznámka: označení stavů a názvy řídicích a stavových signálů na obrázku **neodpovídají** našemu specifickému případu!

Mikroprogramovaný řadič - horizontální



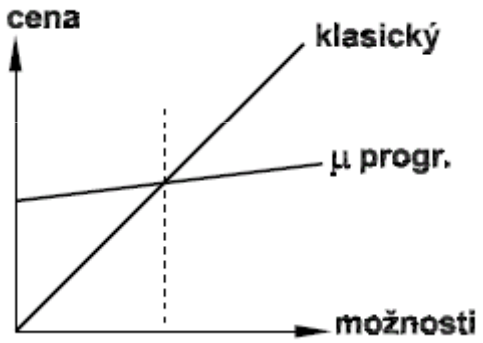
Dovedete si představit, jak by vypadal vertikální, resp. diagonální tvar μOZ (μI)?

Všimněte si:

- Mikroprogramovaný řadič je vlastně počítačem v počítači:
  - RaμI odpovídá PC,
  - Řídicí paměť odpovídá Paměti programu,
  - μOZ odpovídá obsahu IR

Mikroprogramovaný vs. klasický řadič - srovnání

- Rychlost - klasický je rychlejší.
- Cena – levnější je
  - Klasický, ale jen v případě velmi jednoduché variantě.
  - Ve složitější je jím řadič mikroprogramovaný.
- Flexibilita – mikroprogramovaný.
- Změna mikroprogramu – změna chování procesoru.
- Řídicí paměť
  - ROM – pevné mikroinstrukce
  - RWM - µprogramovatelný procesor, možná emulace jiné instrukční sady.

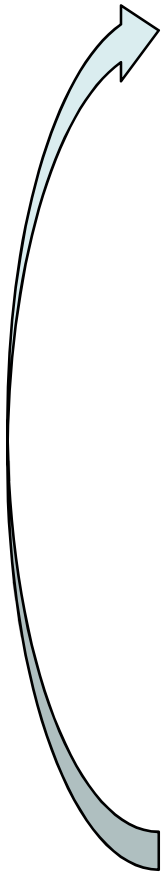


RISC vs. CISC podle Wikipedie

- **RISC** (Reduced Instruction Set Computers), jedna z možných architektur mikroprocesorů.
  - Jde o procesory s redukovanou instrukční sadou optimalizovanou na jejich rychlé vykonávání. Přesná definice není jasná, proto se často tato architektura označuje jako **Load-Store.**
  - Ze společných rysů: instrukce jsou stejně dlouhé, vykonávají se v jednom cyklu. Budeme dále doplňovat.
- **CISC** (Complex Instruction Set Computers), architektura alternativní.
  - Různě dlouhé, různě dlouho trvající instrukce.

    Motorola, Intel x86.

Realistická modifikace základního cyklu počítače



1. Počáteční nastavení, zejména např. PC.
2. Čtení instrukce
   - PC → adresa HP,
   - Čtení obsahu,
   - Přečtená data → IR,
   - PC+*l* → PC, kde *l* je délka instrukce.
3. Dekódování operačního znaku (OZ),
4. provedení operace (včetně vyhodnocení efektivních adres, čtení operandů, apod.).
5. Dotaz na možné přerušení. Ano-li, obsluha.
6. Ne-li, opakování od bodu 2.

Co je přerušení, výjimka?

- Vnější přerušení
  - je metoda pro asynchronní obsluhu vnější události/í. Procesor přeruší sekvenční sémantiku vykonávání instrukcí, přejde na obsluhu. Po jejím ukončení se vrátí na místo, kde přerušení detekoval, a pokračuje v činnosti předchozí.
- Výjimka
  - je přerušením (obsluhou neplánované události) vyvolaným událostí **uvnitř** (v procesoru). Jiné označení pro vnitřní přerušení. -- např. dělení nulou, výpadek stránky, porušení ochrany paměti, …
  - synchronní *softwarová* přerušení - vyvoláno zcela záměrně umístěním příslušné strojové instrukce přímo do prováděného programu - pro vyvolání služeb OS.

Obsluha přerušení

- Uložení adresy místa přerušení a dalších info na zásobník a
- nastavení nového kontextu z vektorů přerušení.