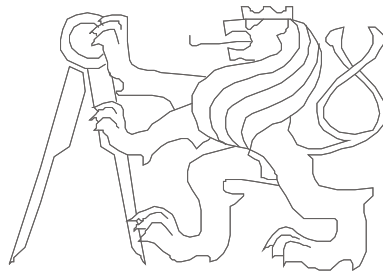


Architektura počítačů

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

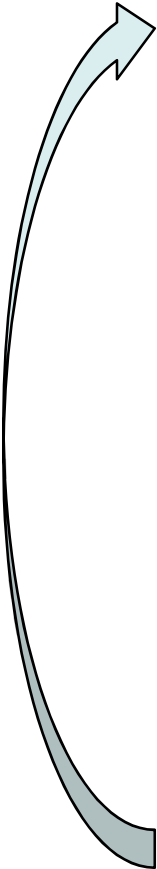
Řadič - součást (jednotka) počítače/procesoru, která jeho činnost řídí.
Sestává ze dvou částí:

- 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 hlavní paměti,
 - Čtení obsahu,
 - Přečtená data → IR,
 - $PC + l \rightarrow PC$, 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.
- 

Kompilace a kódování programu

```
int pow = 1;
int x = 0;

while(pow != 128)
{
    pow = pow*2;
    x = x + 1;
}
```

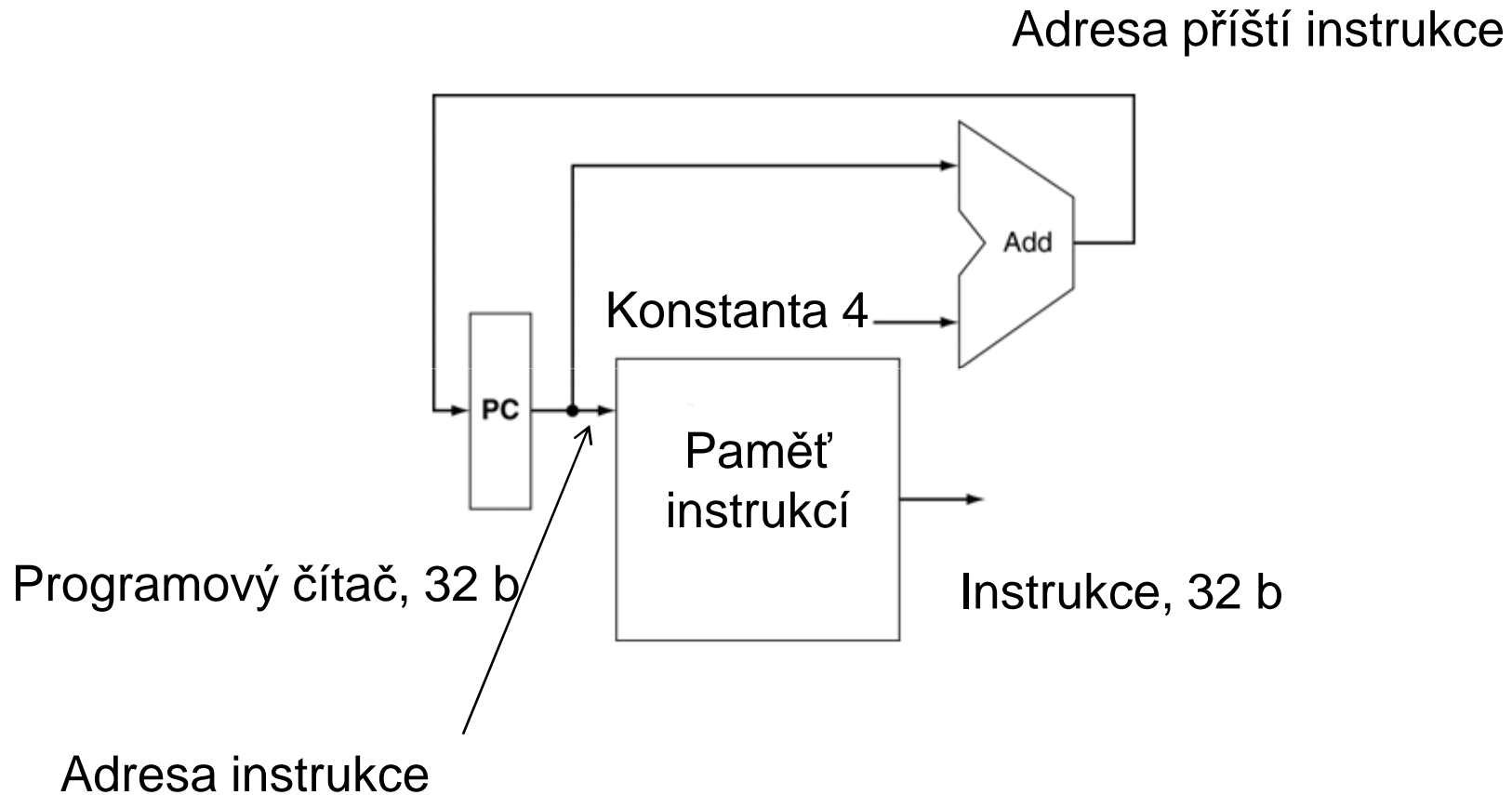
```
addi s0, $0, 1    // pow = 1
addi s1, $0, 0    // x = 0
addi t0, $0, 128  // t0 = 128

while:
    beq s0, t0, done // if pow==128, go to done
    sll s0, s0, 1    // pow = pow*2
    addi s1, s1, 1   // x = x+1
```

j while
done:

8001FFF4	00 00 00 00	NOP
8001FFF8	00 00 00 00	NOP
8001FFFC	00 00 00 00	NOP
80020000	20 10 00 01	start() ADDI \$16, \$00, 0x1
80020004	20 11 00 00	ADDI \$17, \$00, 0x0
80020008	20 08 00 80	ADDI \$08, \$00, 0x80
8002000C	12 08 00 04	while: BEQ \$08, \$16, 0x4
80020010	00 00 00 00	NOP
80020014	00 10 80 40	SLL \$16, \$16, 1
80020018	08 00 80 03	J 0x8003
8002001C	22 31 00 01	ADDI \$17, \$17, 0x1
80020020	00 00 00 00	done: NOP
80020024	00 00 00 00	NOP
80020028	00 00 00 00	NOP

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

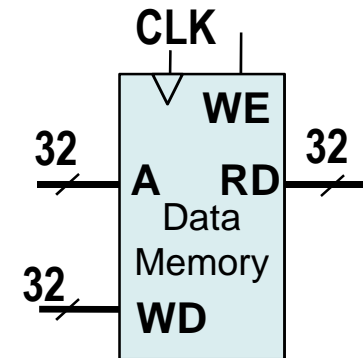
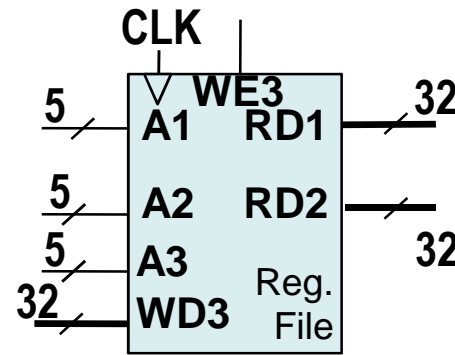
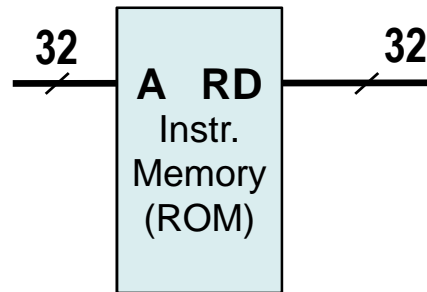
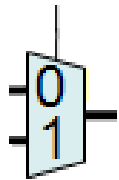
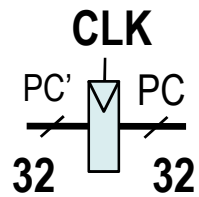
Která je to instrukce?

- Na základe **opcode** a **funct** (pokud je to R typ)

Opcode	
000000	R typ
100011	lw
101011	sw
000100	beq

Funct	
100000	add
100010	sub
100100	and
100101	or
101010	slt

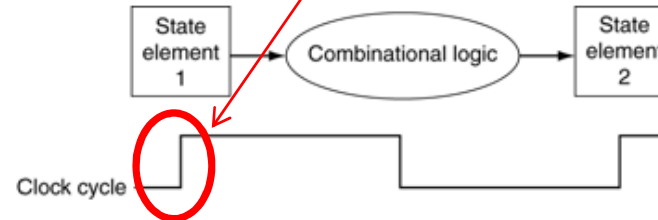
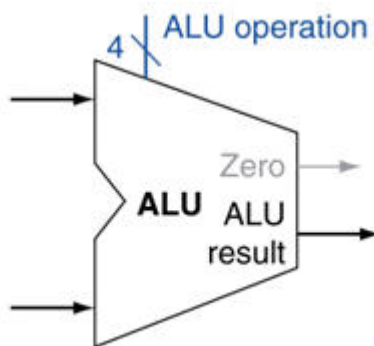
K dispozici máme tyto stavební prvky



Zápis náběžnou hranou CLK při WE = 1

Multiplexor

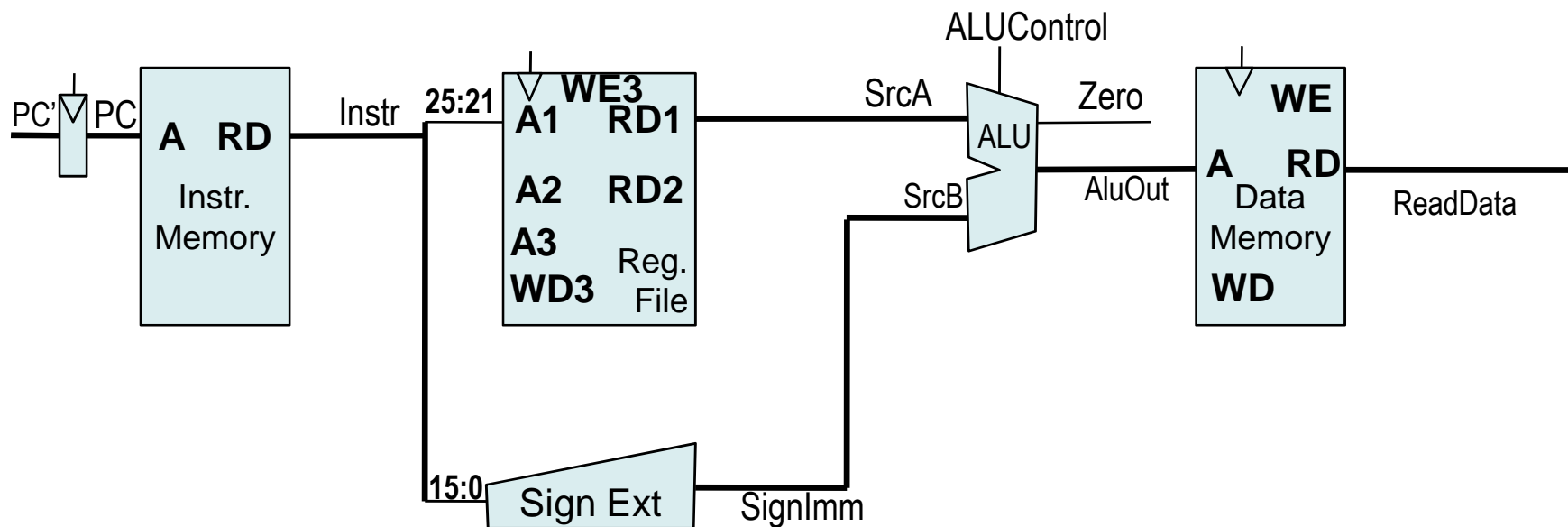
Čtení po uplynutí „dostatečně dlouhé“ doby



Jedno-cyklový procesor – návrh – podpora čtení z paměti

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

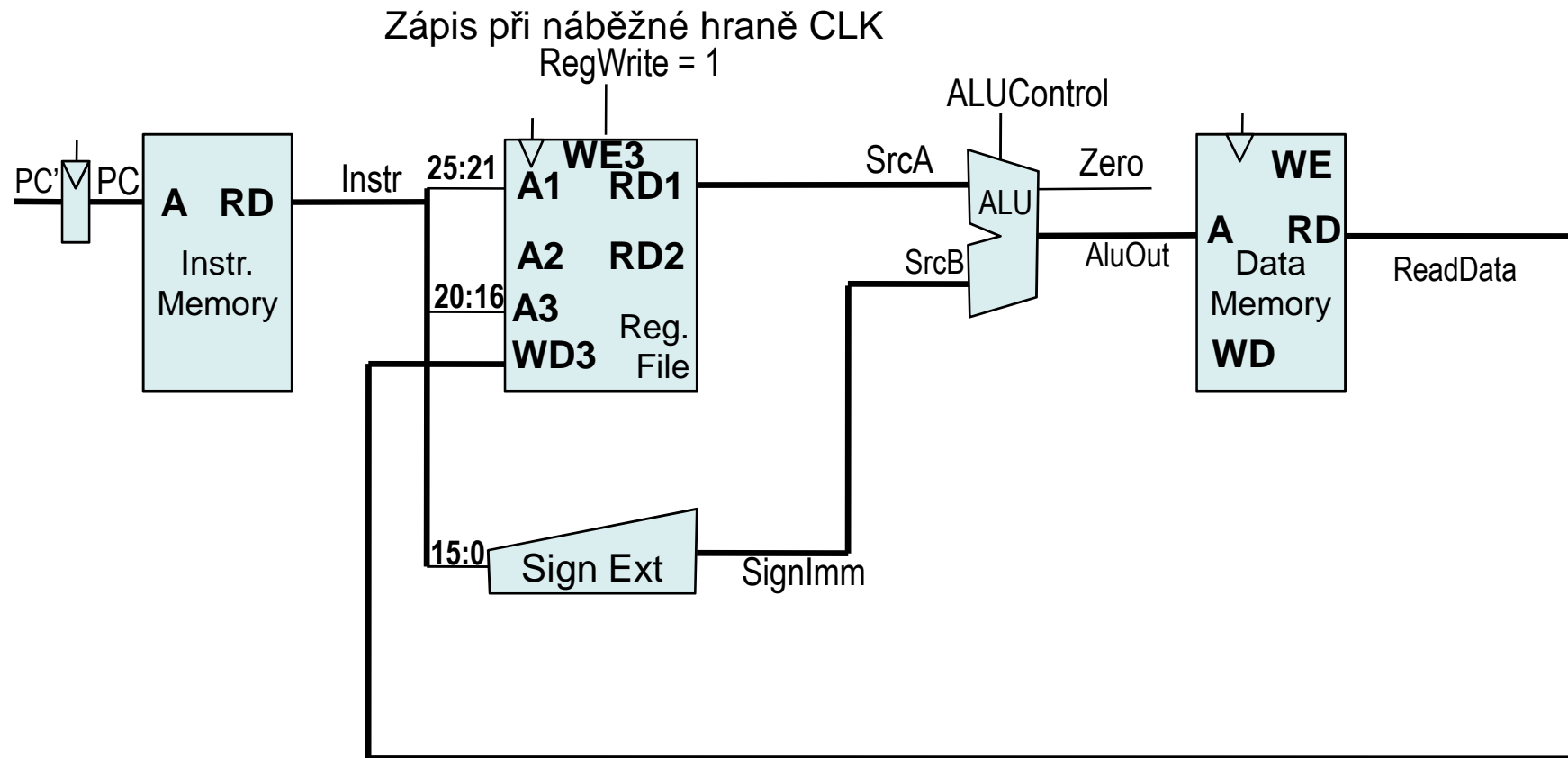
I	opcode (6), 31:26	rs (5), 25:21	rt (5), 20:16	immediate (16), 15:0
---	--------------------------	----------------------	----------------------	-----------------------------



Jedno-cyklový procesor – návrh – podpora čtení z paměti

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

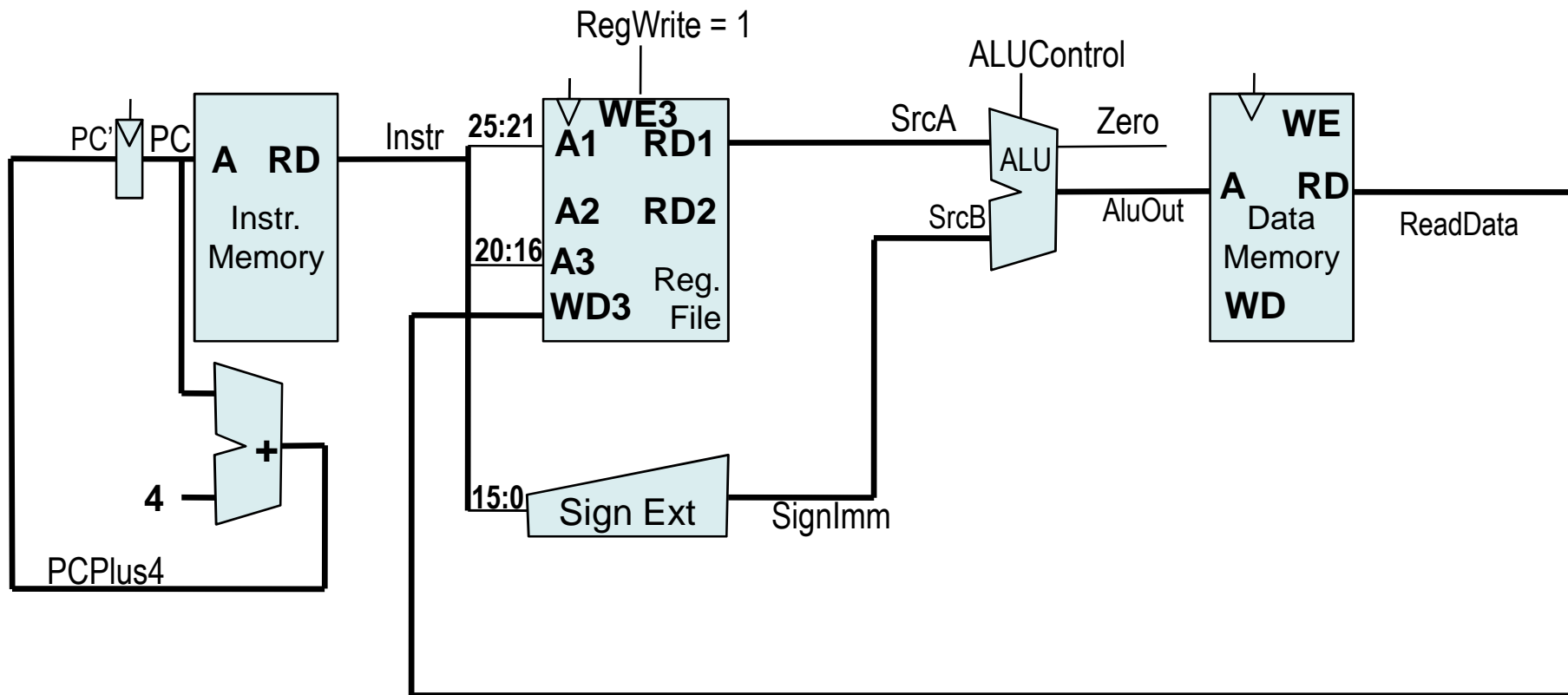
I	opcode (6), 31:26	rs (5), 25:21	rt (5), 20:16	immediate (16), 15:0
---	--------------------------	----------------------	----------------------	-----------------------------



Jedno-cyklový procesor – návrh – podpora čtení z paměti

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

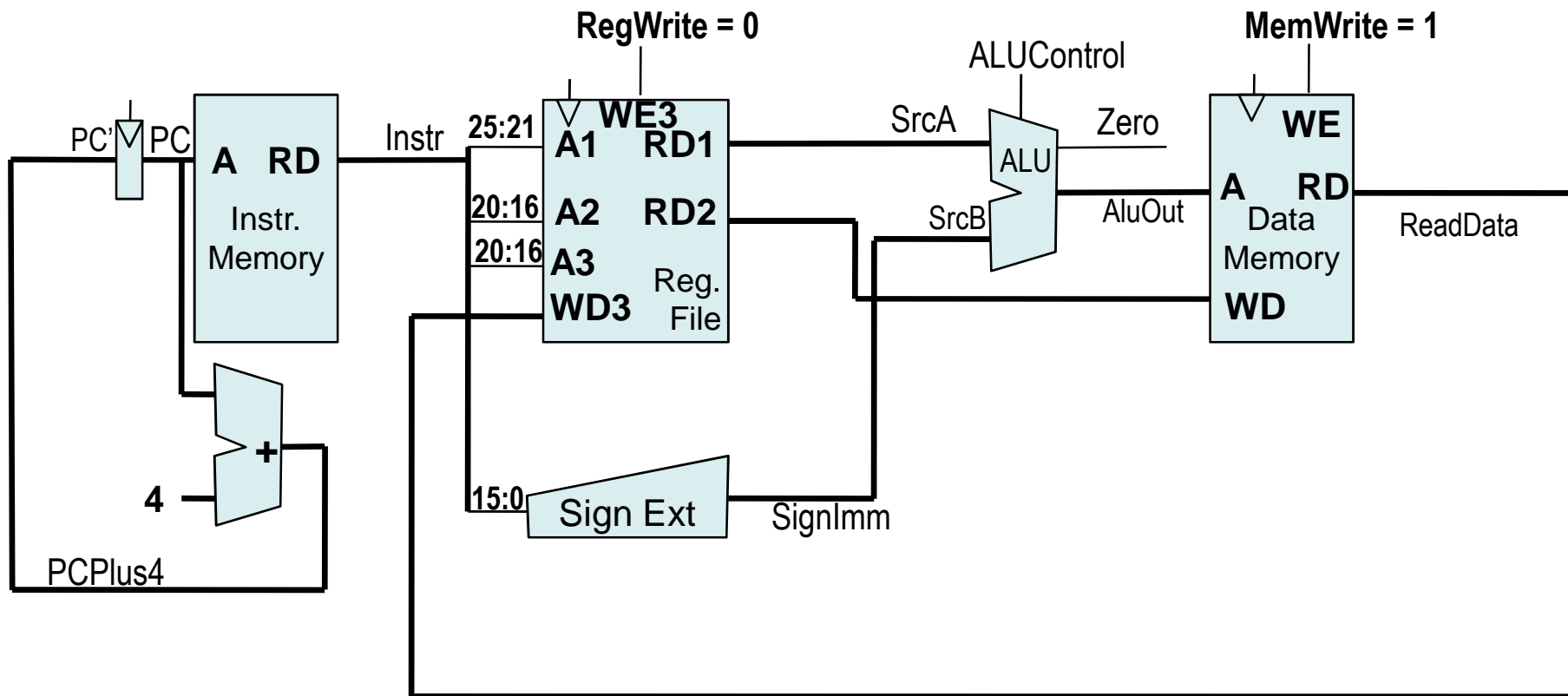
I	opcode (6), 31:26	rs (5), 25:21	rt (5), 20:16	immediate (16), 15:0
---	--------------------------	----------------------	----------------------	-----------------------------



Jedno-cyklový procesor – návrh – podpora zápis do paměti

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

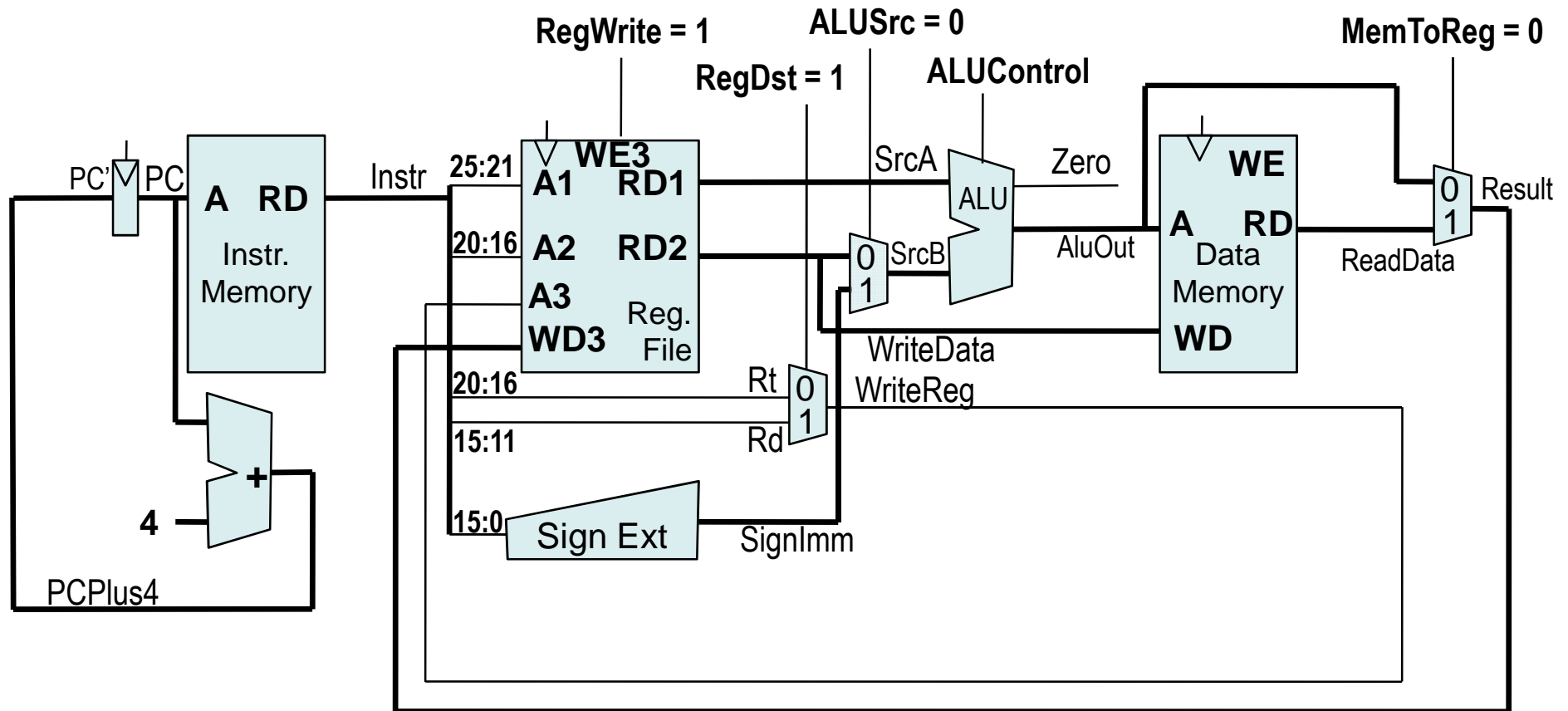
I	opcode (6), 31:26	rs (5), 25:21	rt (5), 20:16	immediate (16), 15:0
---	--------------------------	----------------------	----------------------	-----------------------------



Jedno-cyklový procesor – návrh – podpora add

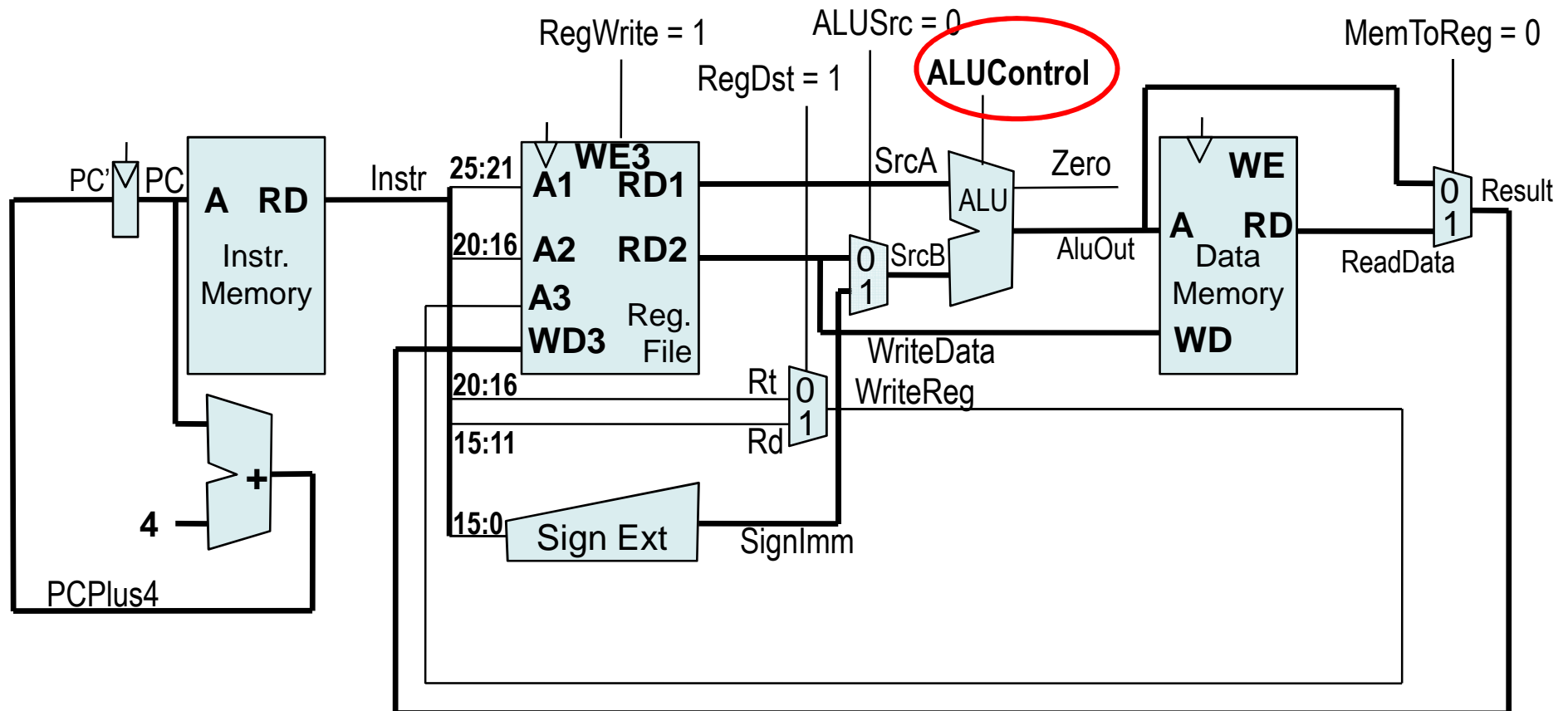
- **add**: typ R; rs, rt – zdroje, rd – cíl, funct – operace součtu

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

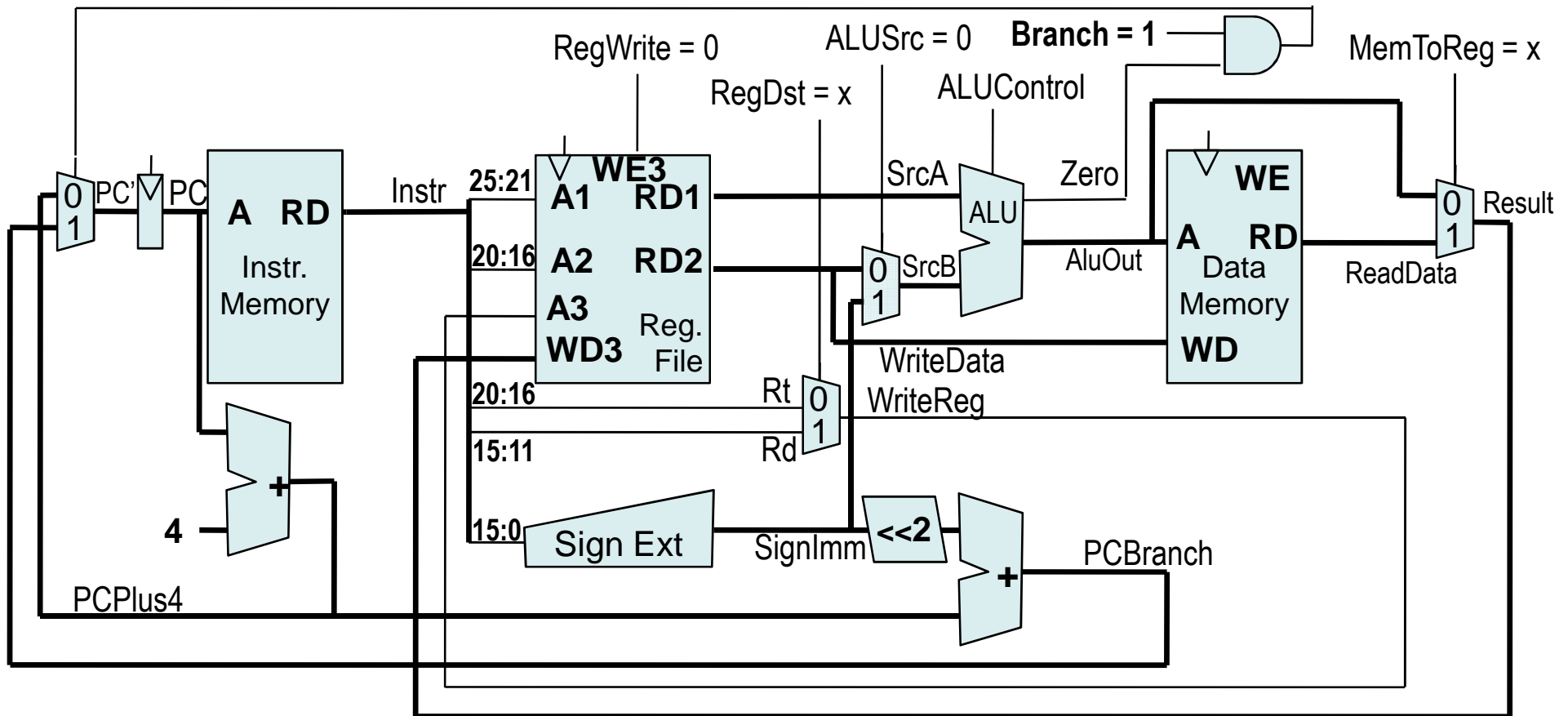
- jediné v čem se liší od add je operace ALU -> datapath beze změny; rozdíl v ALUControl



Jedno-cyklový procesor – návrh – podpora beq

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

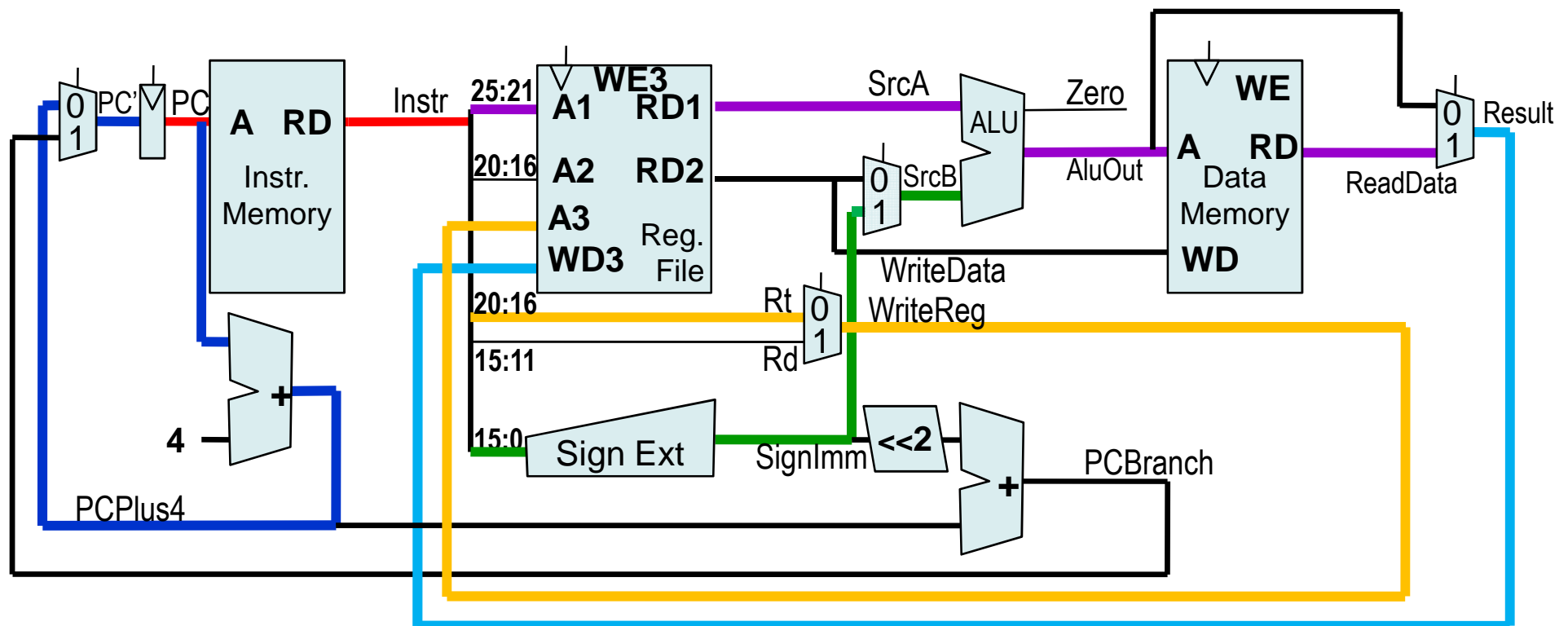
I	opcode(6), 31:26	rs(5), 25:21	rt(5), 20:16	immediate (16), 15:0
---	------------------	--------------	--------------	----------------------



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

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

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



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

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

- Předpokládejme:

$$t_{PC} = 30 \text{ ns}$$

$$t_{Mem} = 300 \text{ ns}$$

$$t_{RFread} = 150 \text{ ns}$$

$$t_{ALU} = 200 \text{ ns}$$

$$t_{Mux} = 20 \text{ ns}$$

$$t_{RFsetup} = 20 \text{ ns}$$

Pak $T_c = 1020 \text{ ns} \rightarrow f_{CLK \max} = 980 \text{ kHz}$,

$IPS = 1.980e3 = 980\,000$ instrukcí za sekundu

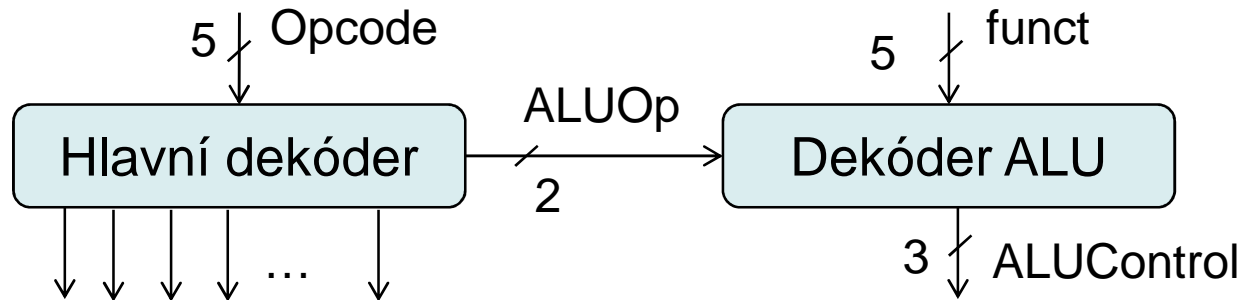
Důležitá poznámka

- Tenhle výsledek si, prosím, zapamatujte.
- Budeme s ním pracovat na 4. přednášce.
- Dnes se dále budeme zabývat porozuměním funkci **řadiče (řídící jednotky)**.

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 základě **opcode** a **funct**



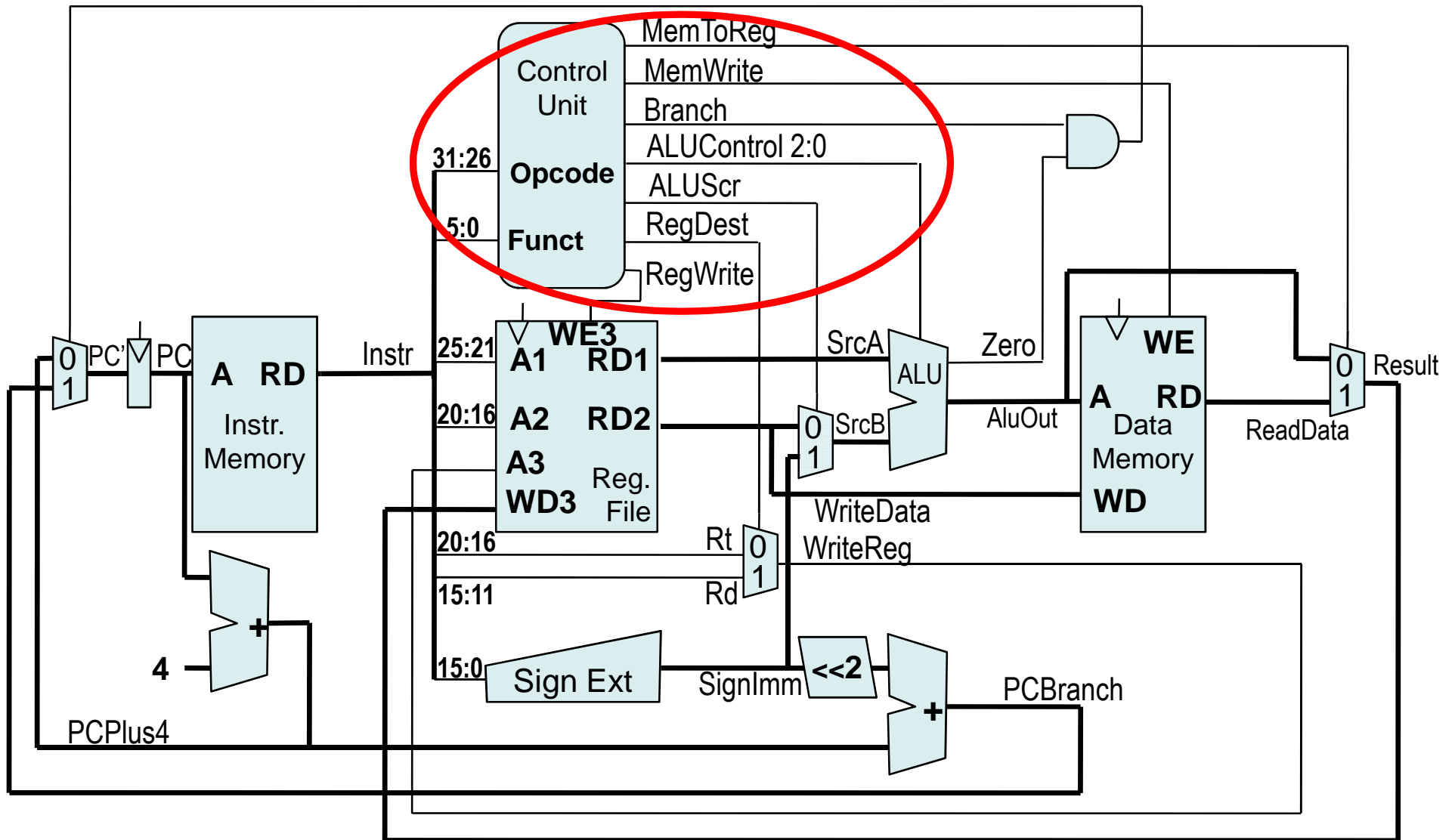
ALUOp	
00	součet
01	rozdíl
10	podle funct
11	-nepoužito-

	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

Řízení ALU (Funkce dekodéru ALU)

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)

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.

Co je řadič procesoru?

Obecněji: Řadič řídí činnost jednotlivých jednotek, koordinuje jejich aktivity a zajišťuje tok informace mezi nimi. Z hlavní paměti získává instrukce (sekvenci instrukcí), které mají být vykonány. Dekóduje je, a na základě typu instrukce nastaví příslušné hradla a datové cesty aby mohla být instrukce vykonána. Obecně, funkcí řadiče je generovat sekvenci řídicích signálů různým subsystémům počítače ve správném pořadí tak, aby byly vykonány požadované operace (aritmetické, změny toku programu aj.). Každý krok v sekvenci kroků vykonávaných řídicí jednotkou v průběhu vykonávání dané instrukce může být definován jako mikro-operace. Mikro-operace je tedy elementární operace (obvykle) vykonaná nad jedním nebo několika registry. Výsledkem mikro-operace je typicky změna obsahu registru (registrů).

Co je řadič procesoru?

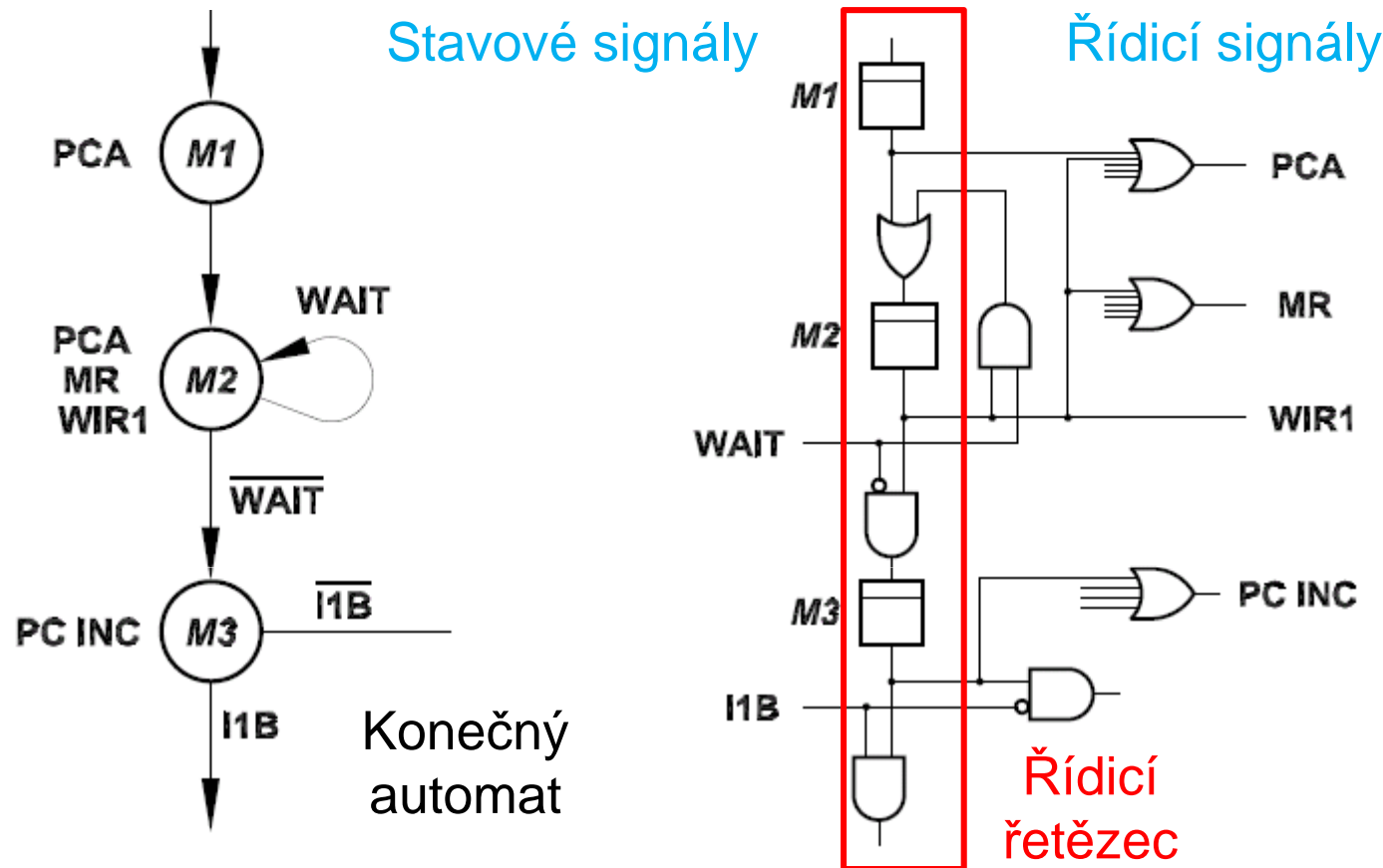
- Příklady mikro-operací:
 - $R(\text{MAR}) \leftarrow R(\text{CIAC})$
Obsah Current Instruction Address Counter do Memory Address Register
 - $R(\text{CIAC}) \leftarrow R(\text{CIAC}) + 1$
Inkrementace registru CIAC
 - $M(\text{MBR}) \leftarrow M(\text{MAR})$
Výběr z paměti
 - IF F(S) THEN $R(\text{A}) \leftarrow R(\text{MDR})$

K realizaci mikro-operace může být zapotřebí jednoho nebo několika řídicích signálů.

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.

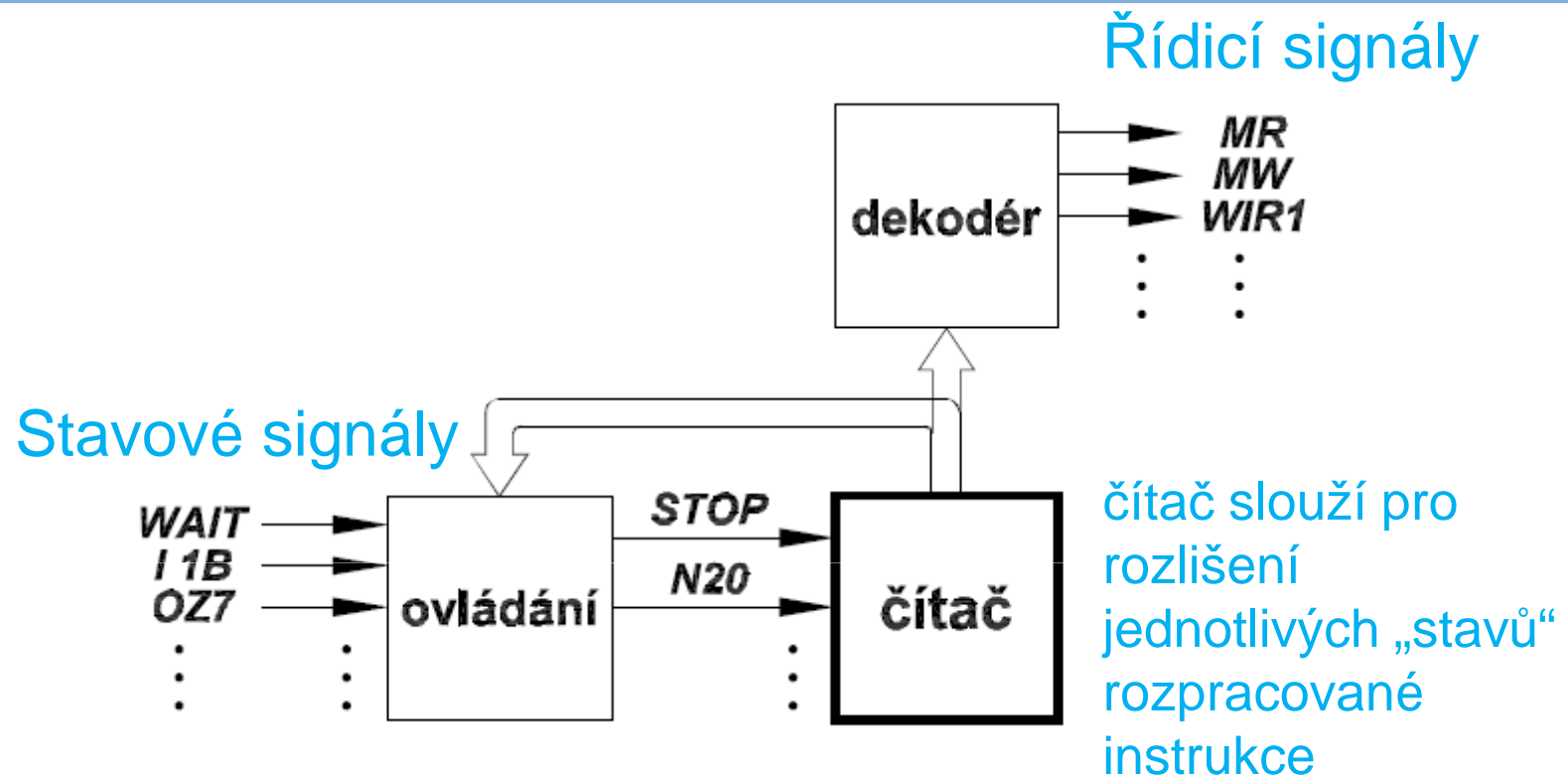
Realizace: řadič s řídicími řetězci



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

Činnost řadiče může být popsána konečným automatem (lze relativně popsat jazykem na popis HW – Verilog/VHDL), zejména při multicyklovém vykonávání instrukcí. Jednotlivým stavům automatu přináležejí konkrétní nastavení řídicích signálů, přechodům pak podmínky (stavová hlášení, typ instrukce,...), při kterých se mezi těmito stavy přechází.

Řadič na bázi čítače



„Ovládání“ mj. nastavuje a zastavuje čítač (když vyhodnotí konec operace). Úkolem „dekodéru“ je nastavit řídicí signály pro vykonávací subsystém v závislosti od stavu čítače.

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

Mikroprogramovaný řadič

Nedílnou součástí *mikroprogramovaného řadiče* je *řídící paměť* obsahující *mikroinstrukce*, přičemž každá ze strojových instrukcí je provedena pomocí jedné nebo několika jednodušších mikroinstrukcí.

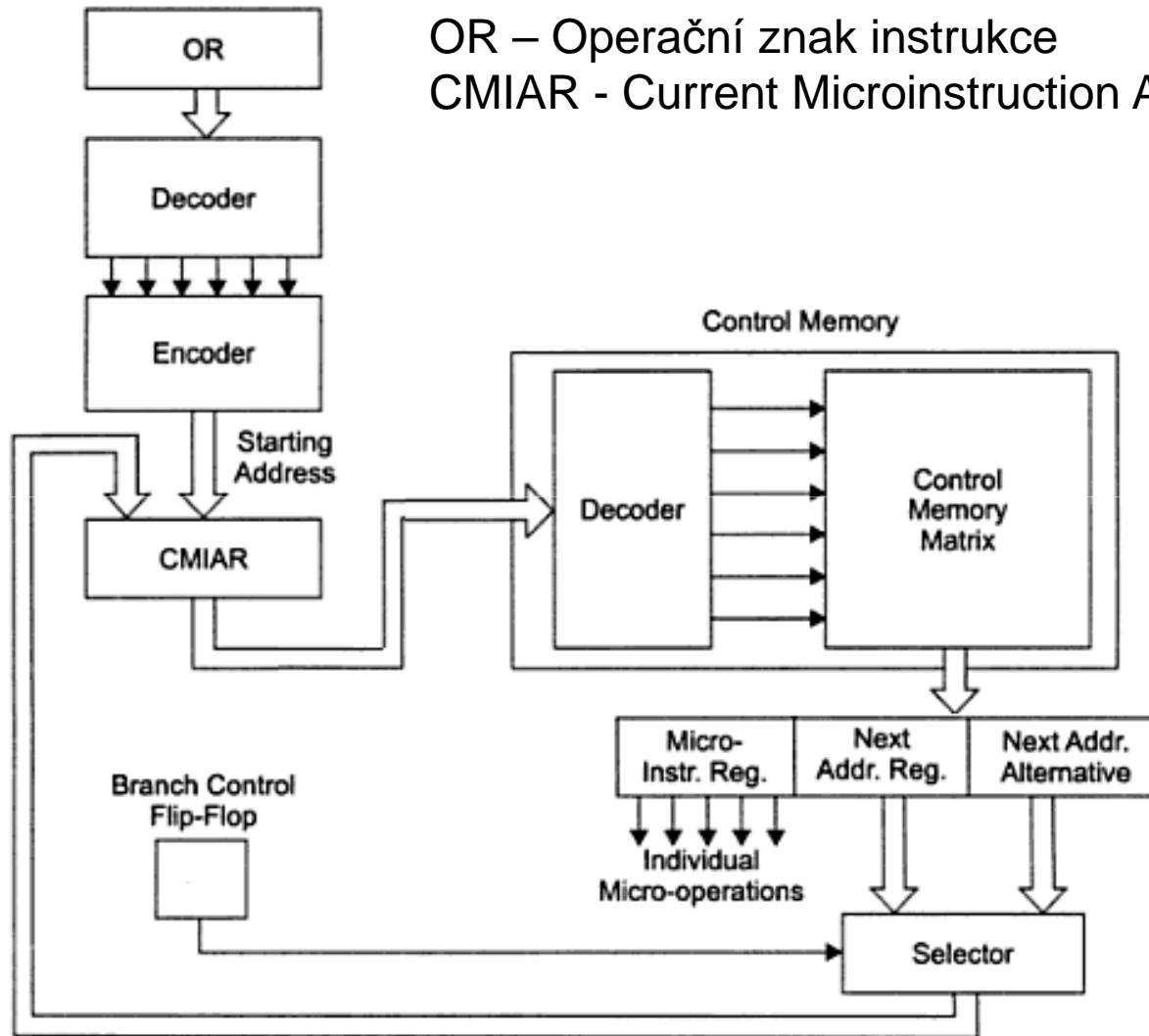
Vstup - kódy instrukcí načtené z operační paměti

Výstup - řídicí signály uvnitř procesorou (ALU, interní sběrnice...), externí signály (paměť,...)

Operační znak instrukce udává adresu první mikroinstrukce z řídicí paměti, která implementuje danou operaci.

Mikroprogram implementuje programátorovi viditelné strojové instrukce (add, sub, lw, xor, jmp...) - ISA

Mikroprogramovaný řadič



OR – Operační znak instrukce

CMIAR - Current Microinstruction Address Register

Mikroprogramovaný řadič

Organizace mikroinstrukcí – horizontální, vertikální a diagonální

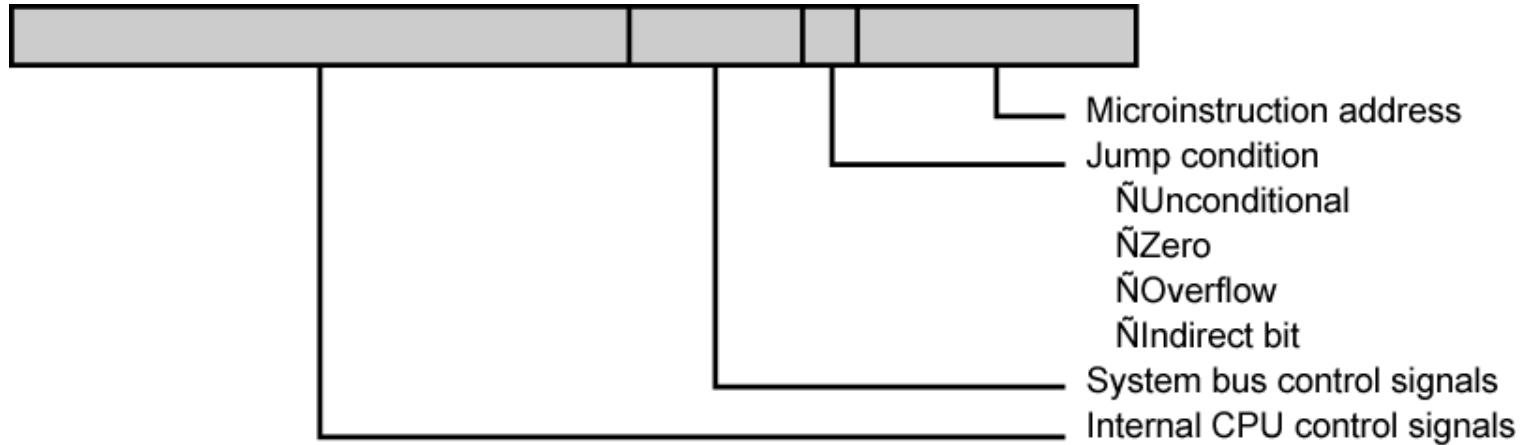
Vertikální:

- Pokud budeme uvažovat N typů mikrooperací, postačuje $\log_2 N$ bitů pro specifikaci mikrooperace. Nicméně potřebujeme dekodéry pro generování řídicích signálů. Jedna mikroinstrukce pak vykoná pouze jednu mikrooperaci. Kódování po skupinách -> Jedna mikroinstrukce pak může vykonat několik mikrooperací.

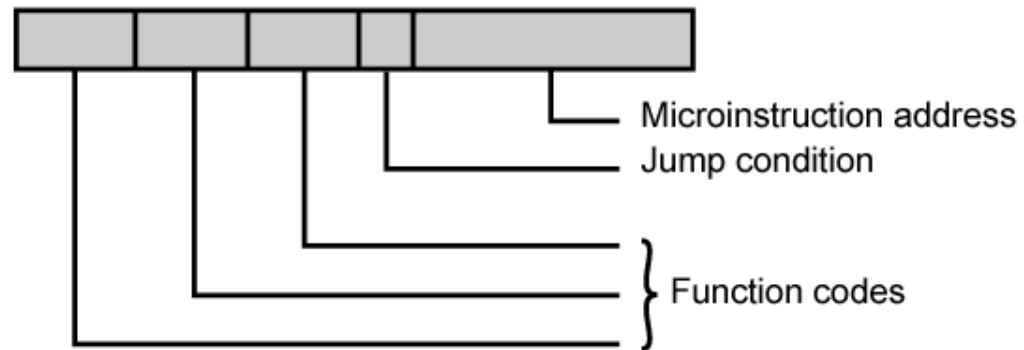
Horizontální:

- Každý bit mikroinstrukce může reprezentovat specifickou mikrooperaci. Takže pro N typů mikroinstrukcí potřebujeme N bitů. Můžeme vykonat libovolnou množinu mikrooperací paralelně v jediné mikroinstrukci.

Mikroprogramovaný řadič



(a) Horizontal microinstruction



(b) Vertical microinstruction

Mikroprogramovaný řadič

Diagonální:

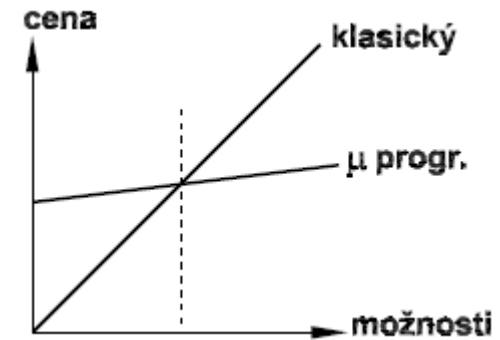
Kompromis. Některé bloky zakódovány „horizontálně“ a jiné „vertikálně“. Horizontální formát má bity pro jednotlivé signály přímo uloženy v mikroinstrukci (bez nutnosti dalšího dekódování). Vertikální formát kóduje větší skupinu navzájem se vylučujících signálů (aktivní je pouze jeden) společně do jednoho bloku.

Poznámka k terminologii: **Horizontální** vs. **vertikální** primárně slouží k rozlišení toho, zda mikroinstrukce přímo řídí součásti CPU (horizontální), nebo potřebuje další dekódovací stupně (vertikální).

Mikroprogramovaný řadič je vlastně počítačem v počítači.

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 - μ programovatelný procesor, možná emulace jiné instrukční sady.
- Mikroprogramovaný řadič – neefektivní pro zřetězené procesory (alespoň jako centralizovaný) – každý stupeň vykonává jinou instrukci. Bylo by potřeba zajistit správné provedení všech instrukcí napříč stupni spolu s řešením hazardů.



Mikroprogramovaný řadič

Závěr k mikroprogramovaným řadičům:
Mikroprogram je vlastně vrstvou mezi strojovými instrukcemi a samotným HW. Slouží k implementaci strojových instrukcí v řídicí jednotce CPU, GPU, řadičů disků, řadičů síťových rozhraní, atd. Pomáhá oddělit strojové instrukce od elektronických obvodů, což rovněž pomáhá implementovat komplexní instrukce bez nárůstu složitosti HW. Programování pomocí mikroinstrukcí se označuje mikroprogramování. Napsaný kód, který je uložen (ROM, PLA, flash) uvnitř řadiče je pak mikroprogram. Při návrhu řadičů moderních procesorů využívajících zřetězení použití mikroprogramovaného řadiče není populární, mj. i z důvodu samotné podstaty sekvenčního vykonání mikroprogramu.