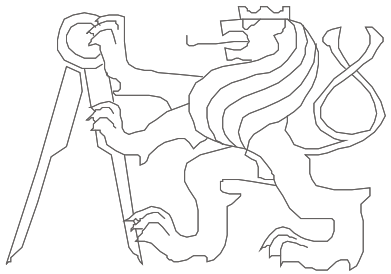
Architektura počítačů

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:

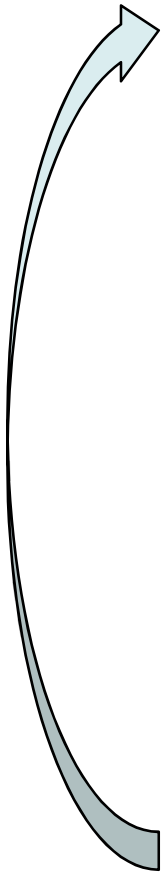
- datové
  - registry,
  - další potřebné obvody,
- vlastní řídicí části, z tzv. jádra řadiče.

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

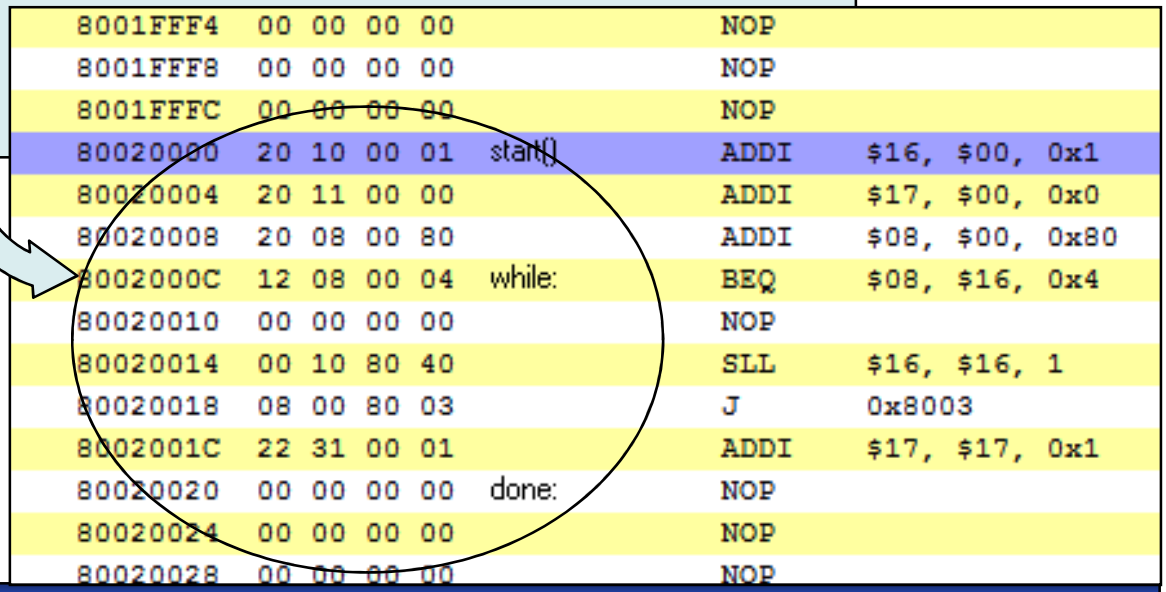
1. Počáteční nastavení, zejména např. PC.
2. Čtení instrukce
   - PC → adresa hlavní paměti,
   - Čtení obsahu,
   - Přečtená data → IR,
   - PC+*l* → PC, kde *l* je délka instrukce.
3. Dekódování operačního znaku (OZ),
4. provedení operace (včetně vyhodnocení efektivních adres, čtení operandů, apod.).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



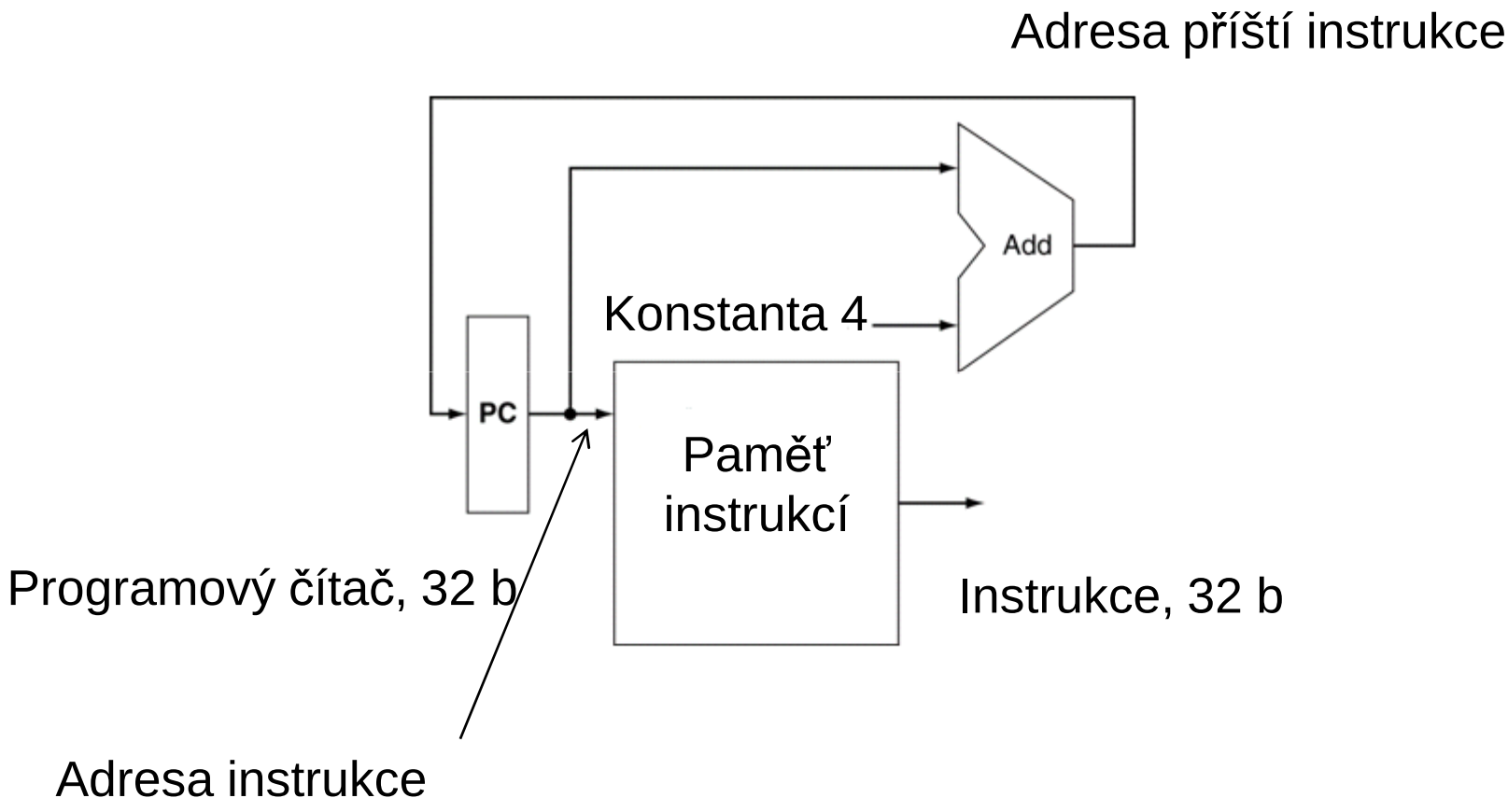
Kompilace a kódování programu

```
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:
```



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

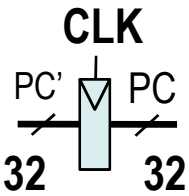
Která je to instrukce?

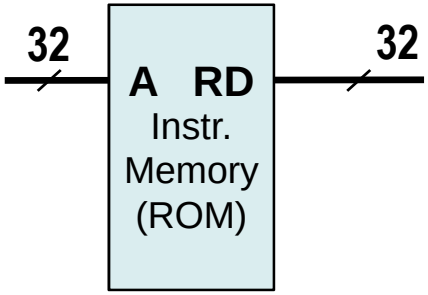
- Na základě **opcode** a **funct** (pokud je to R typ)

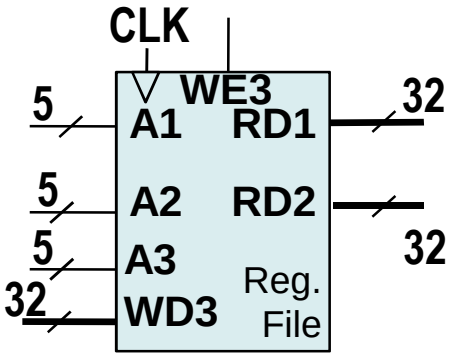
Opcode	
000000	R typ
100011	lw
101011	sw
000100	beq

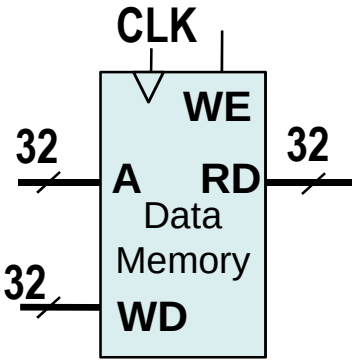
Funct	
100000	add
100010	sub
100100	and
100101	or
101010	slt

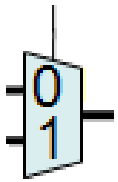
K dispozici máme tyto stavební prvky







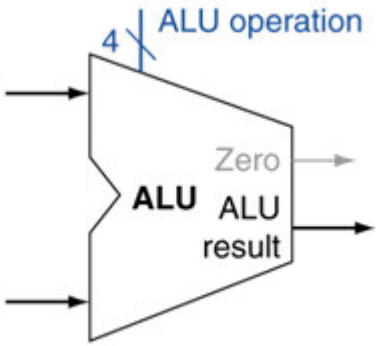


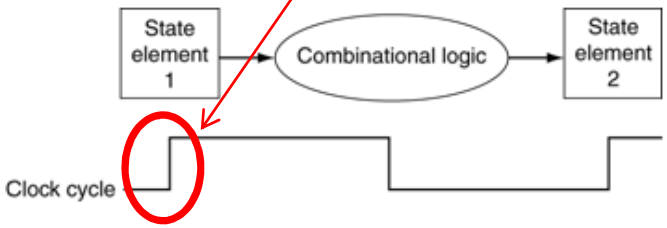


Multiplexor

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

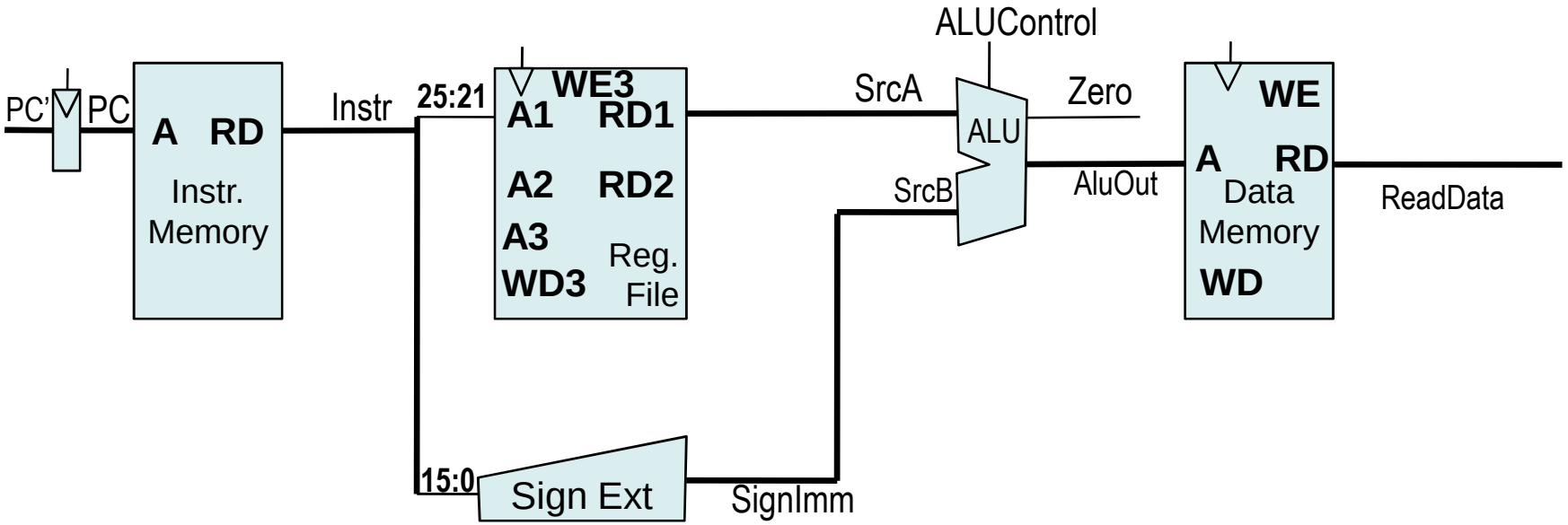
0x 8C 0B 00 04 – strojový kód instrukce lw $11, 0x4($0)

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

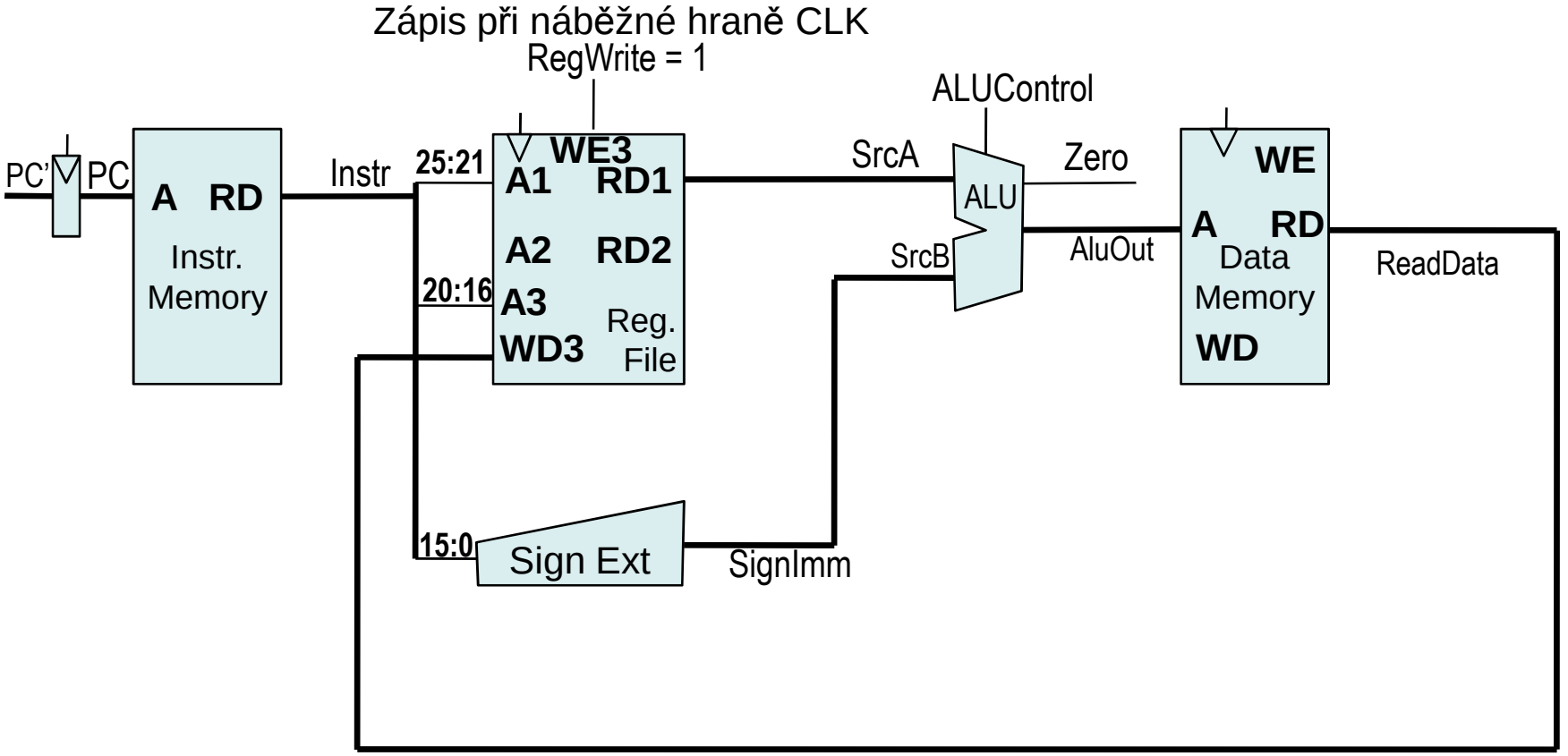
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora čtení z paměti

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

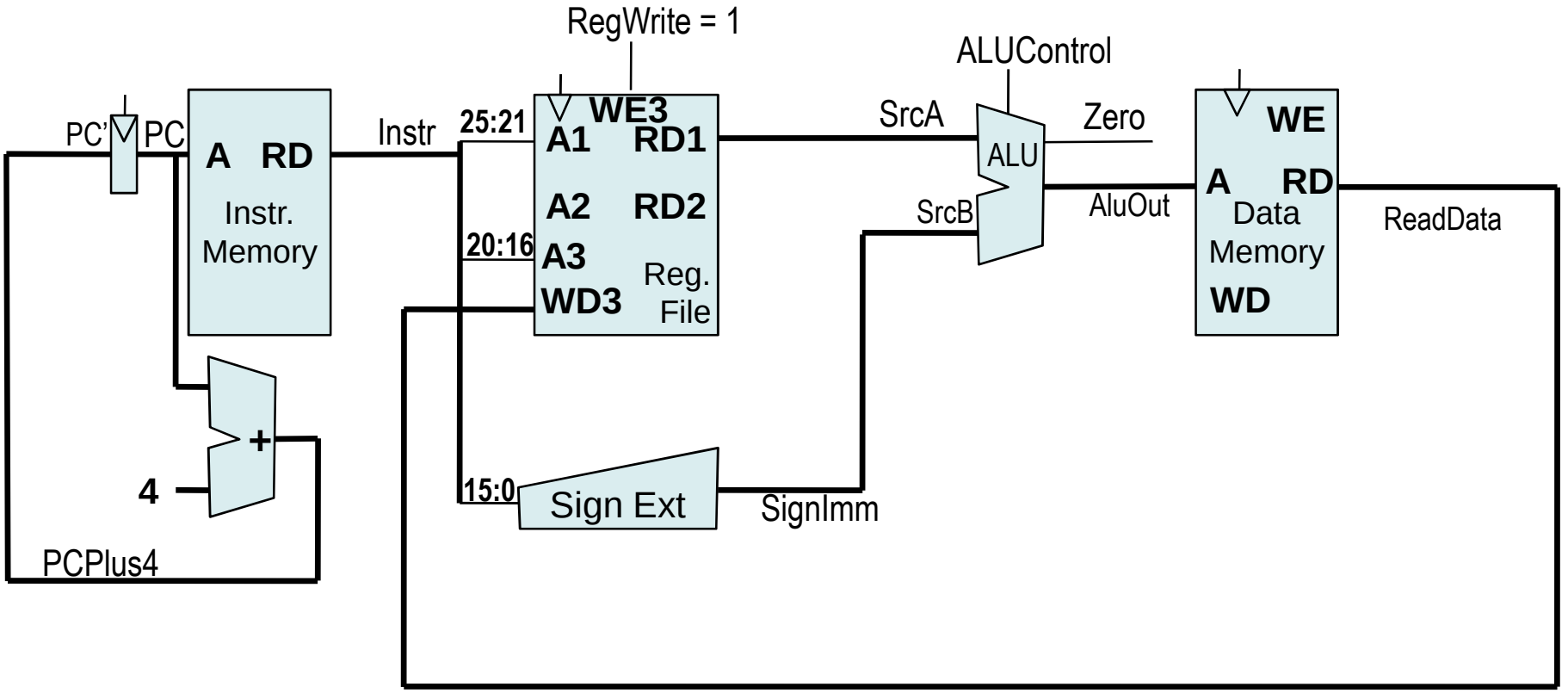
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora čtení z paměti

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

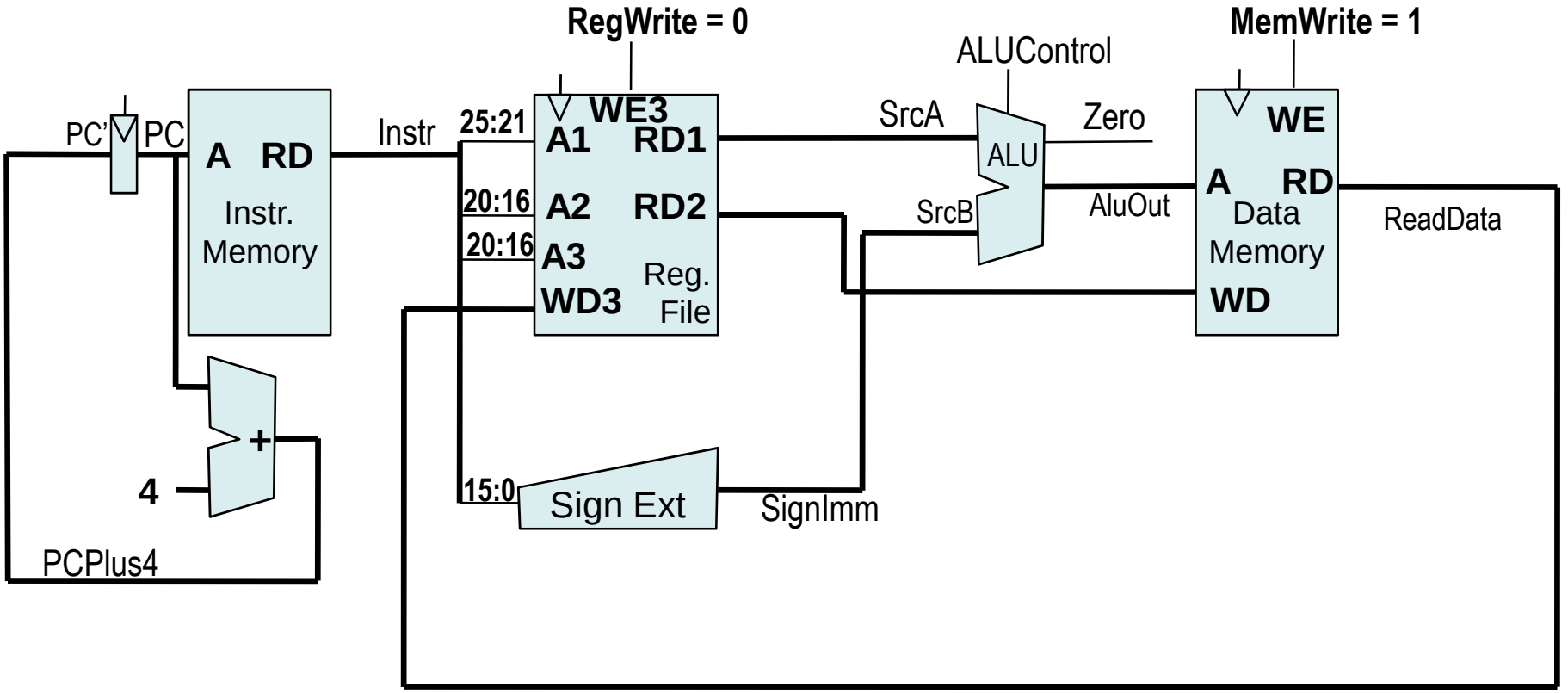
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora zápis do paměti

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

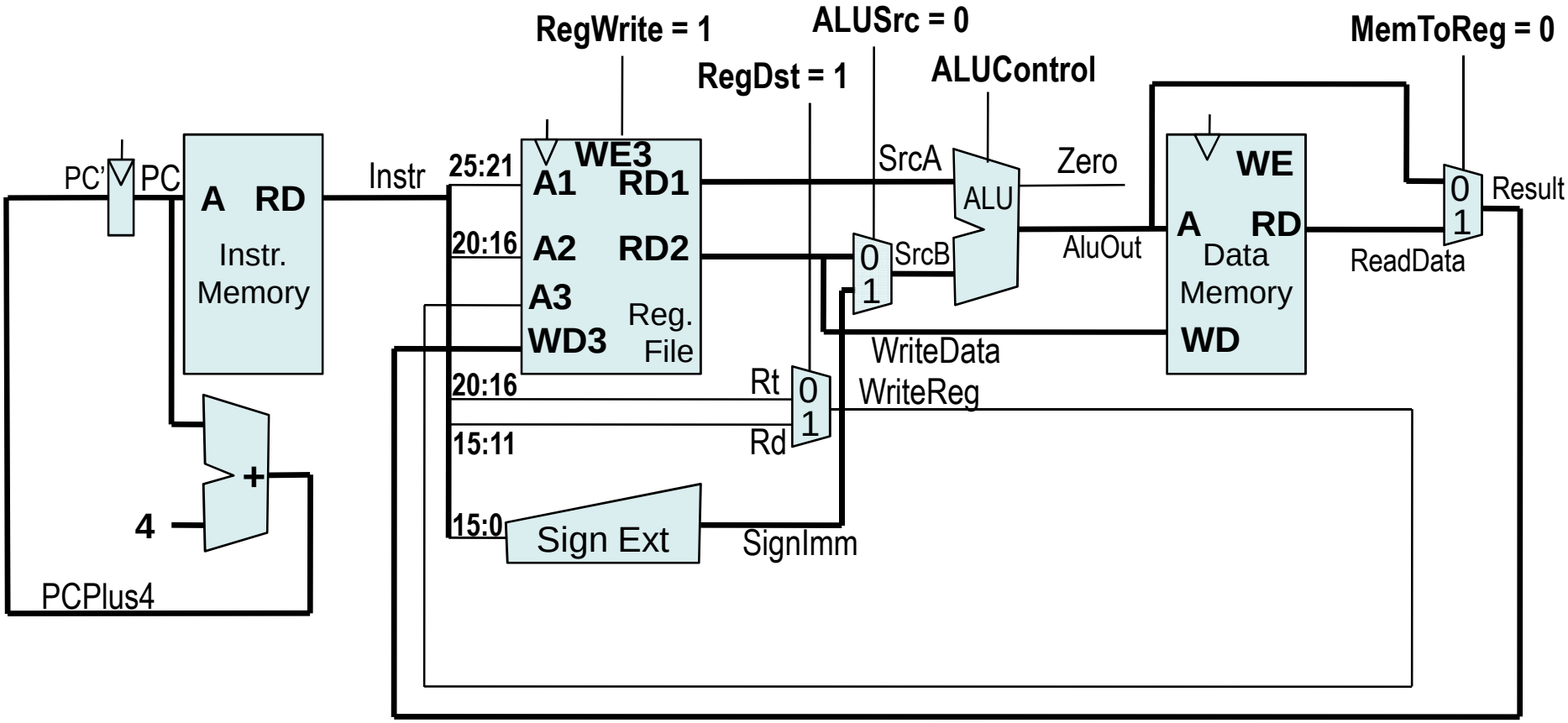
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora add

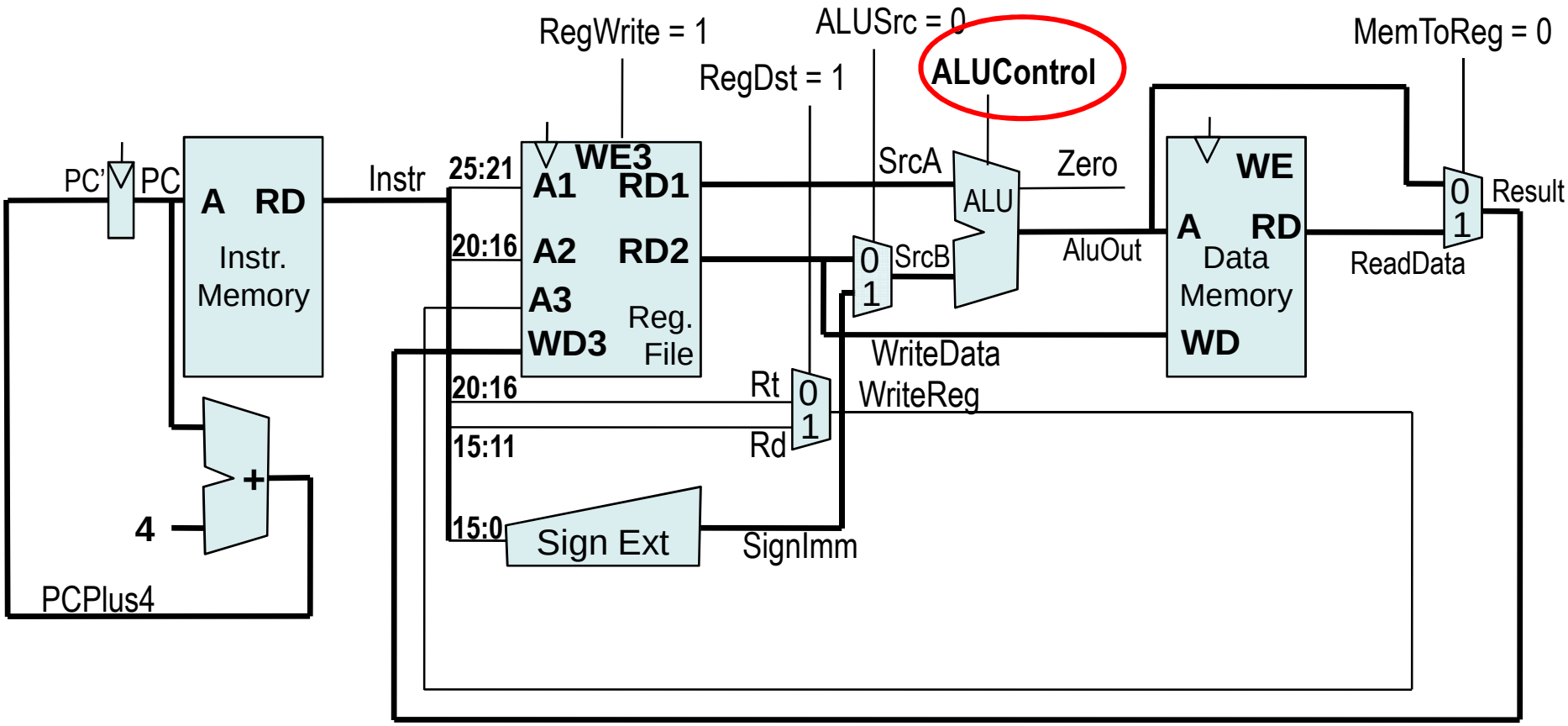
- **add**: typ R; rs, rt – zdroje, rd – cíl, funct – operace součtu

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`

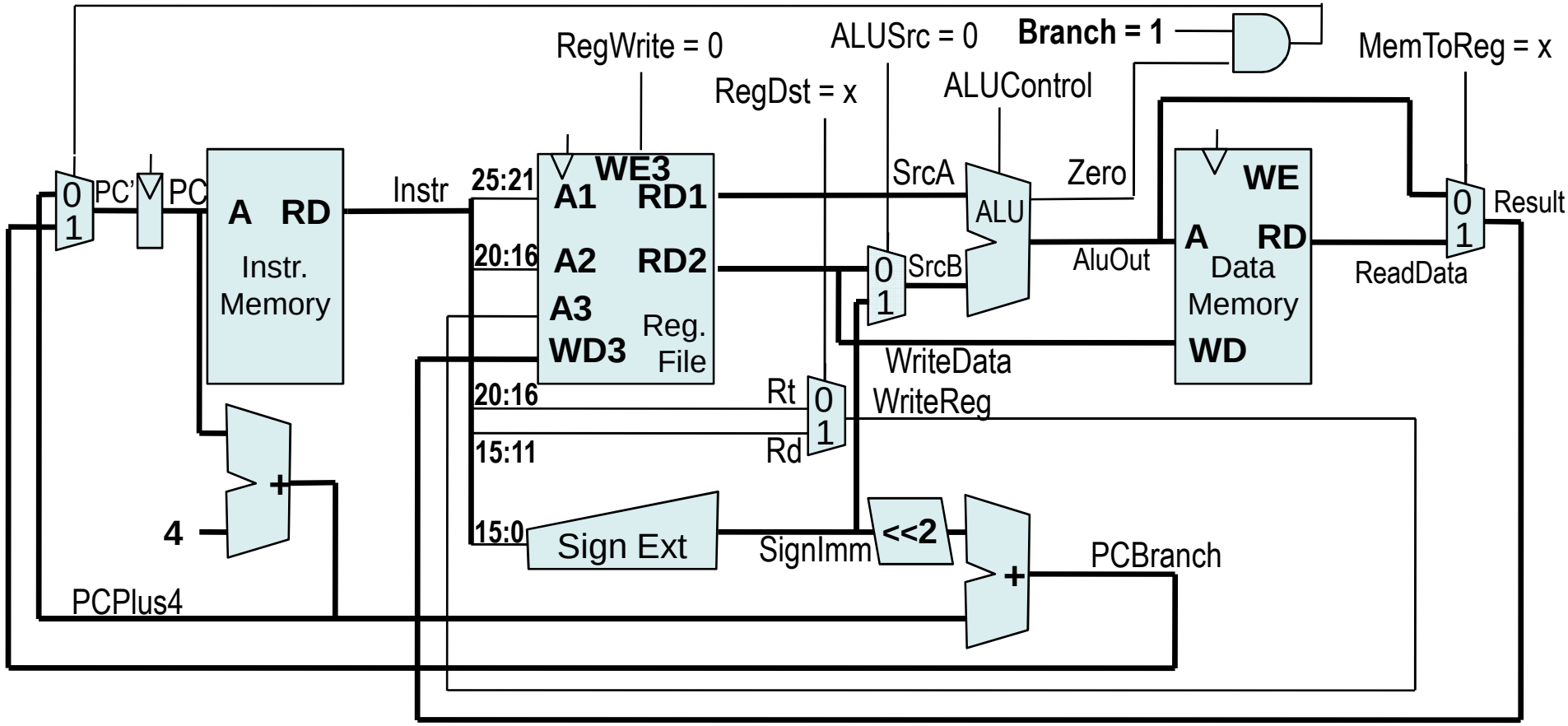
- jediné v čem se liší od `add` je operace ALU -> datapath beze změny; rozdíl v ALUControl



Jedno-cyklový procesor – návrh – podpora beq

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

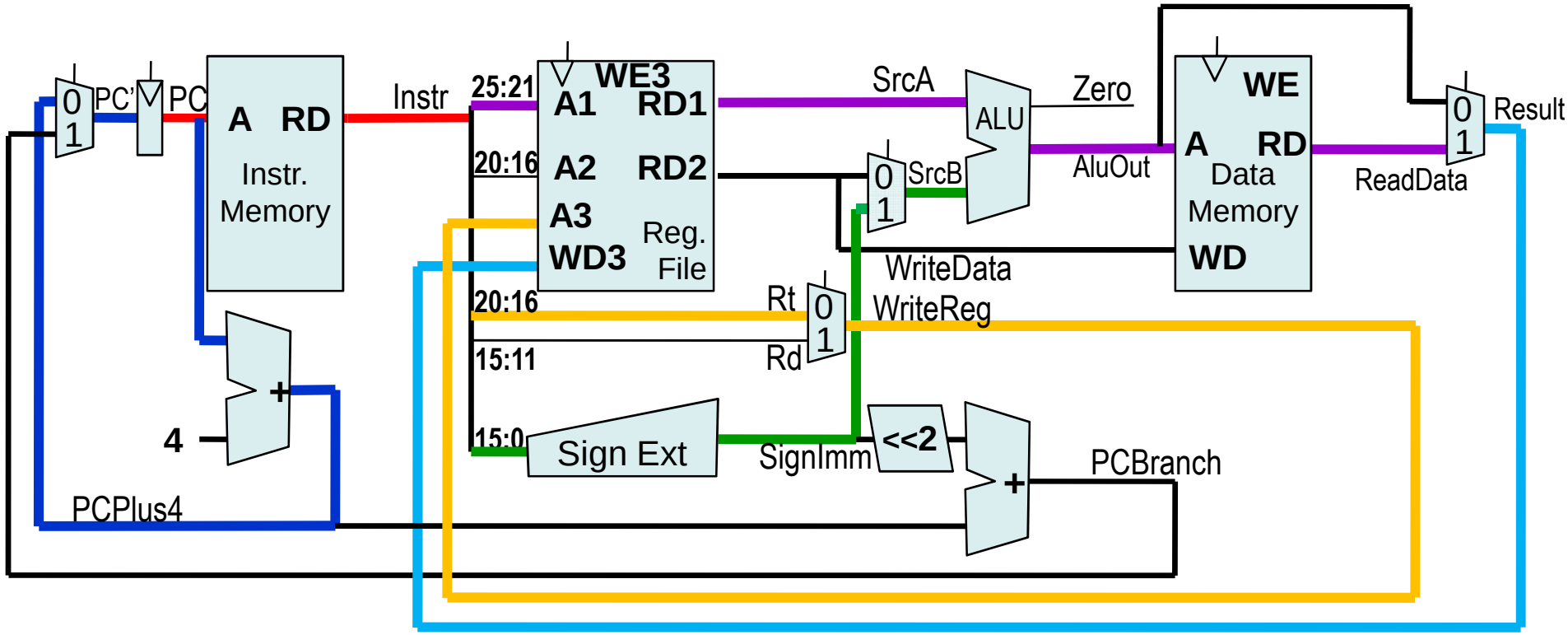
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



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

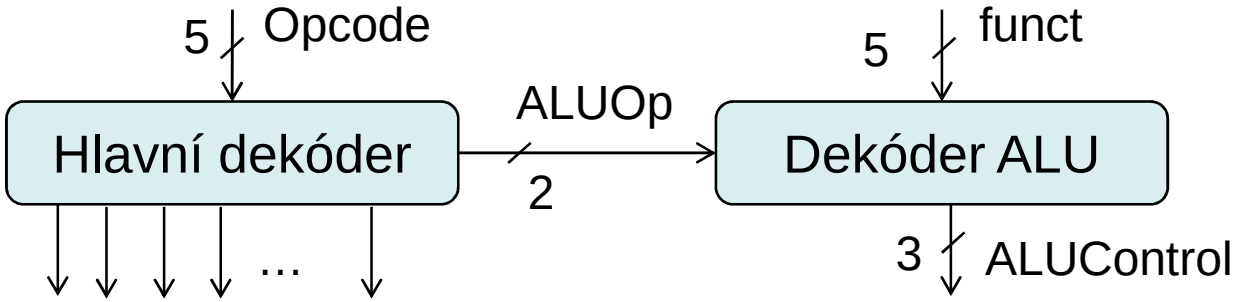
- Tenhle výsledek si, prosím, zapamatujte.
- Budeme s ním pracovat na 4. přednášce.

- Dnes se dále budeme zabývat porozuměním funkci **řadiče (řídicí 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 základe **opcode** a **funct**



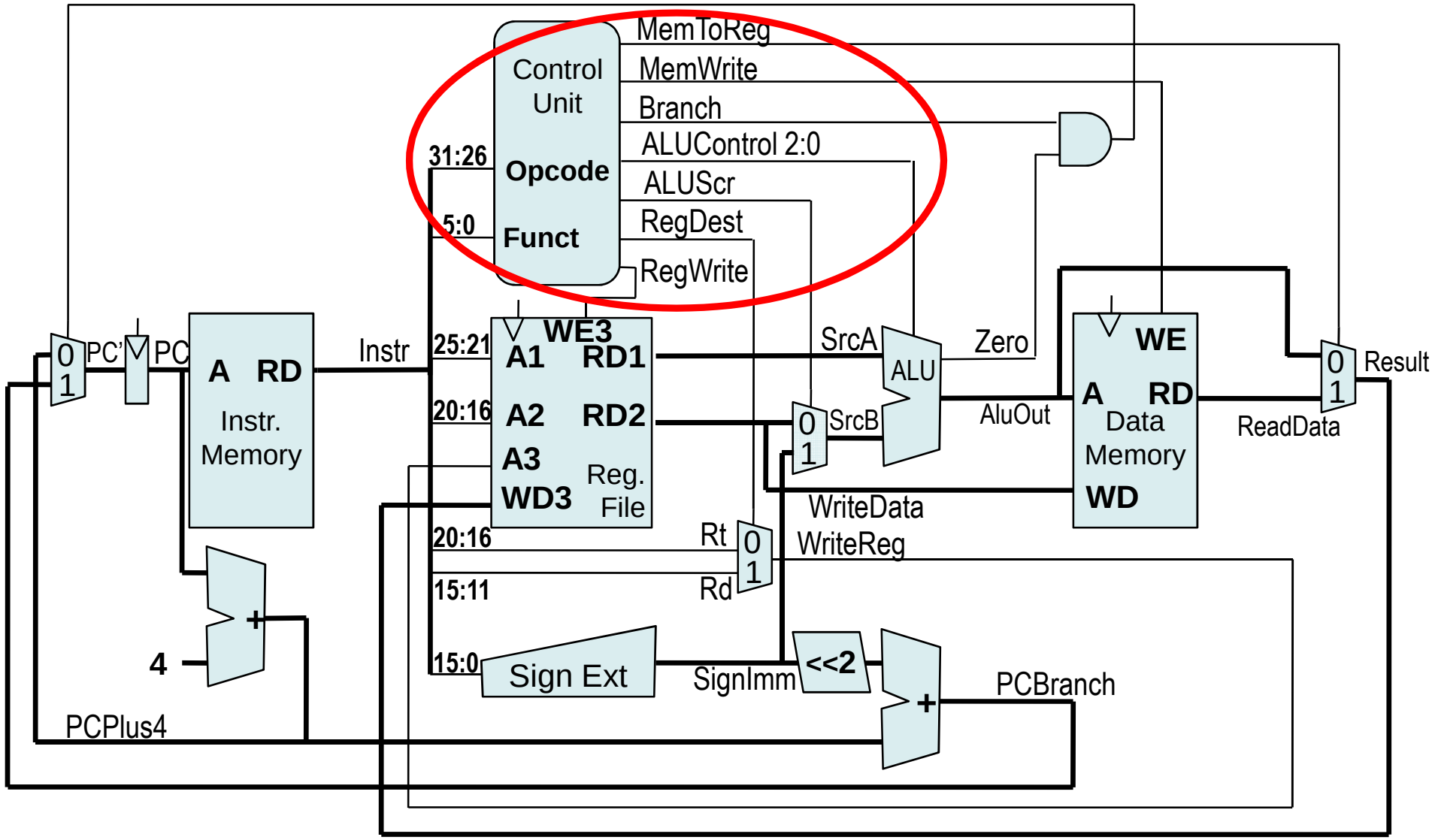
ALUOp	
00	součet
01	rozdíl
10	podle funct
11	-nepoužito-

	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

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 (řídicí jednotka) 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.

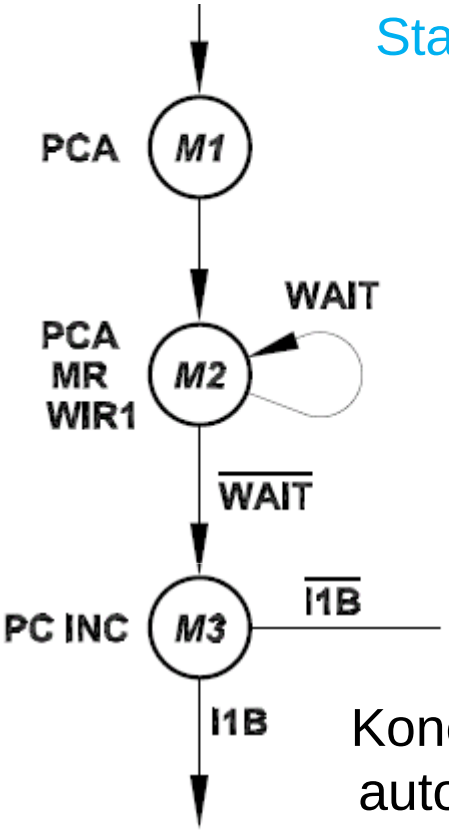
Co je řadič procesoru?

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 signálu 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).

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.

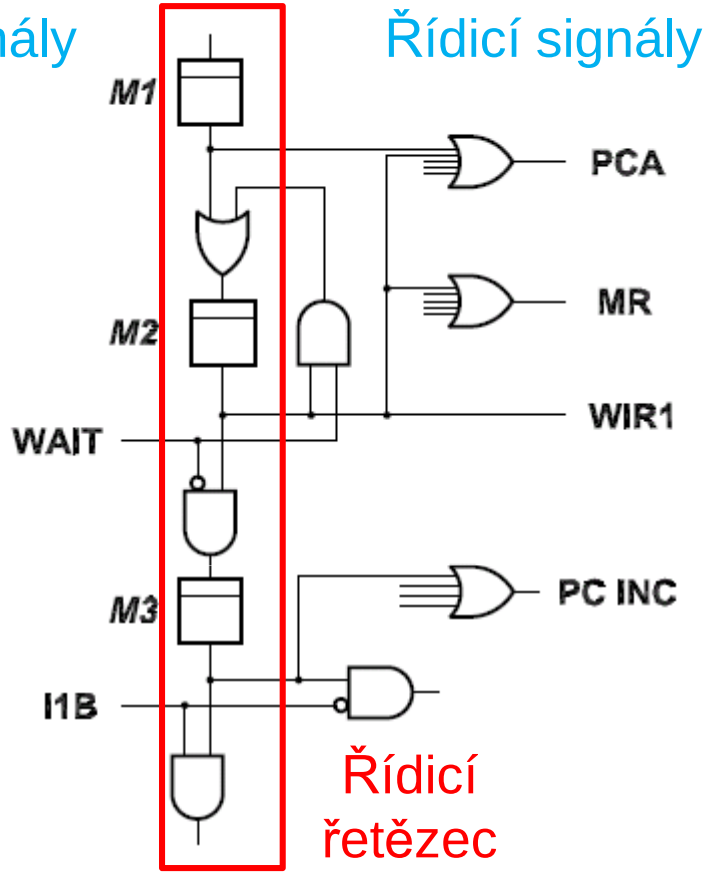
Realizace: řadič s řídicími řetězci



Konečný automat

Stavové signály

Řídicí signály

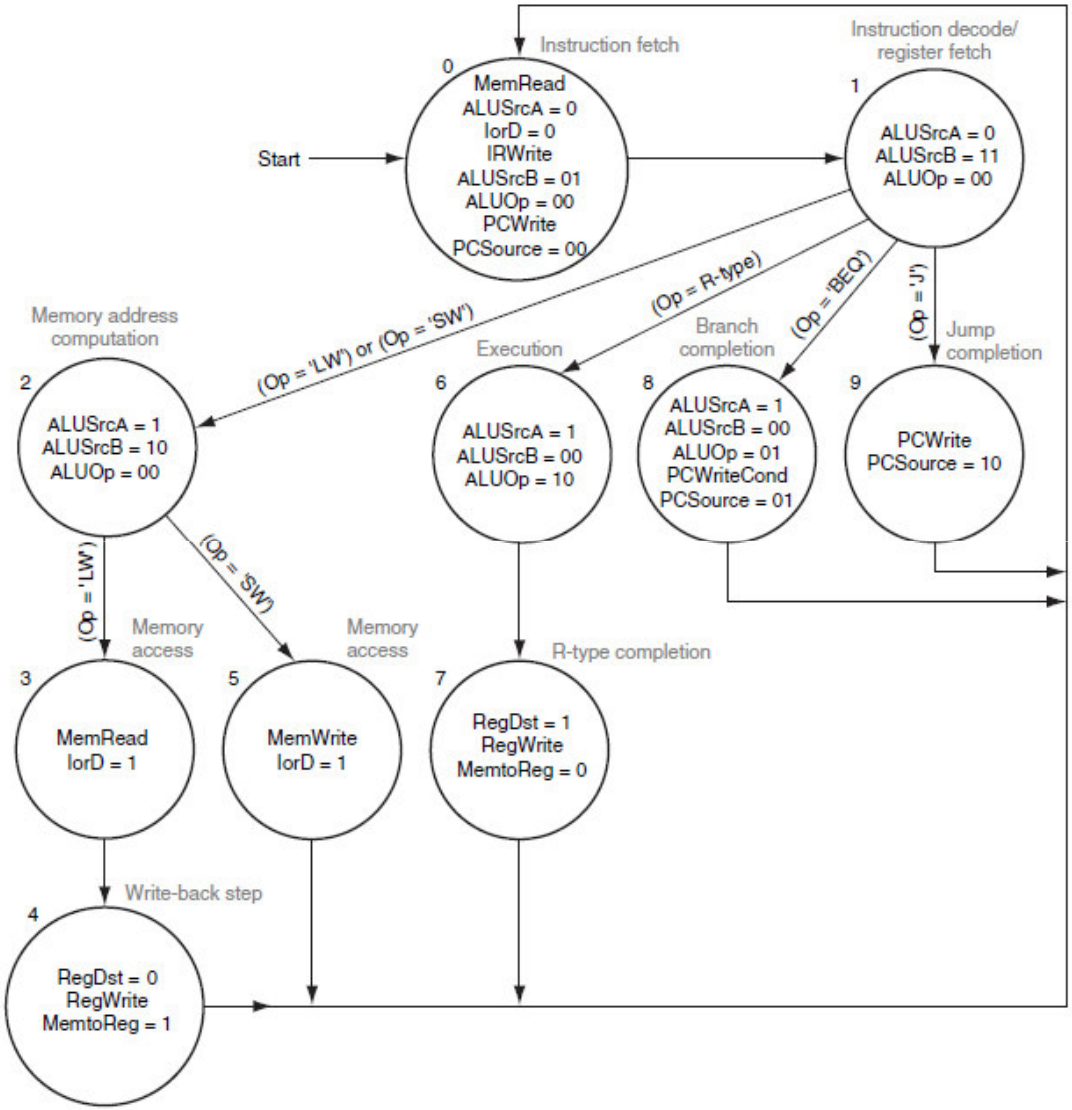


Řídicí řetězec

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 snadno popsat jazykem na popis HW – Verilog/VHDL), zejména při multicyklovém vykonávání instrukcí. Jednotlivým stavům automatu přináleží konkrétní nastavení řídicích signálů, přechodům pak podmínky (stavová hlášení, typ instrukce,...), při kterých se mezi těmito stavy přechází.

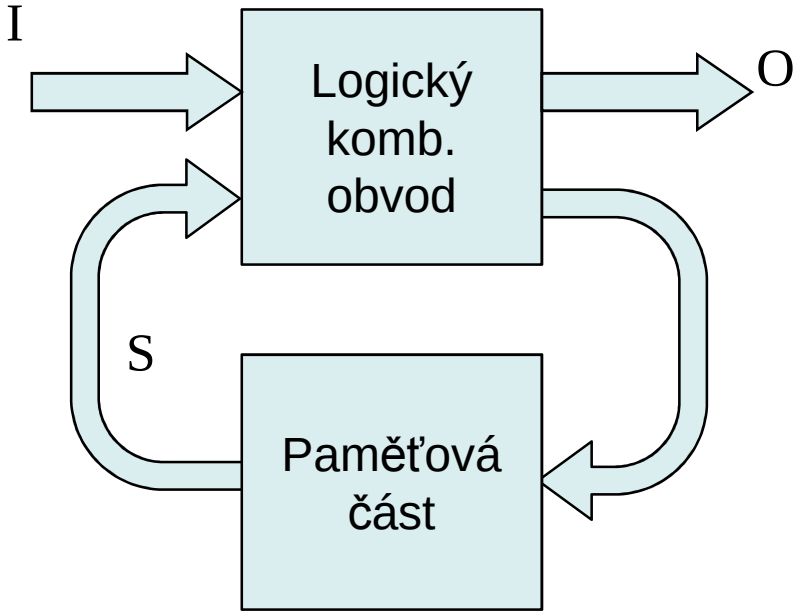
Stavový automat řadiče - příklad



Stavový automat řadiče - příklad

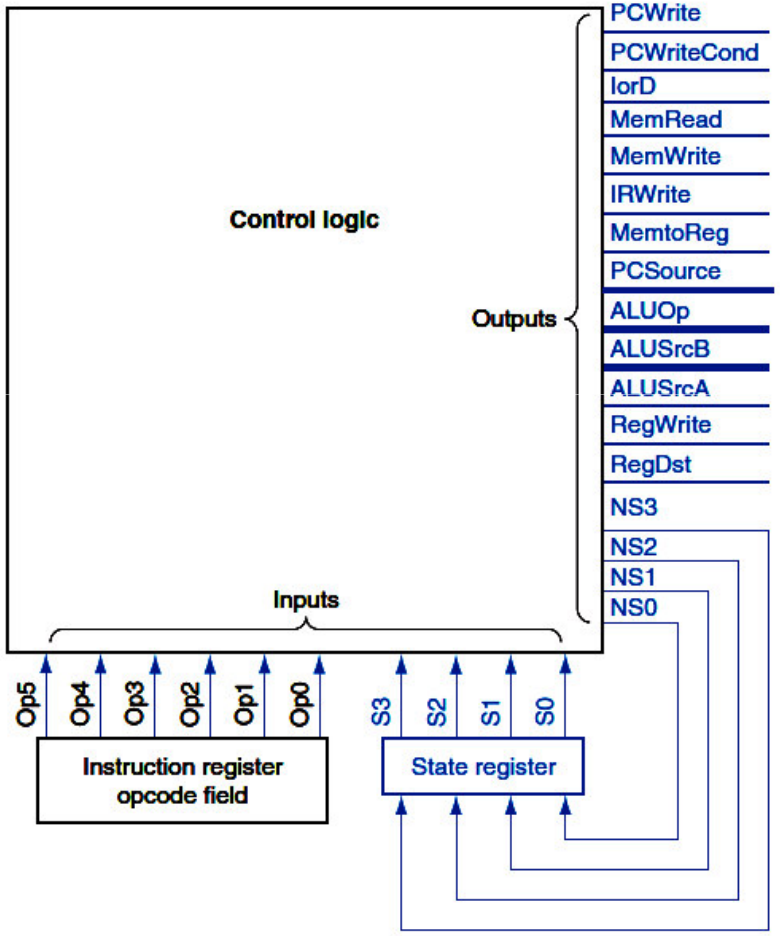
Obecný model logického sekvenčního obvodu (Huffmann)

$O^{t+1}=g(S^t, I^t)$



$S^{t+1}=f(S^t, I^t)$

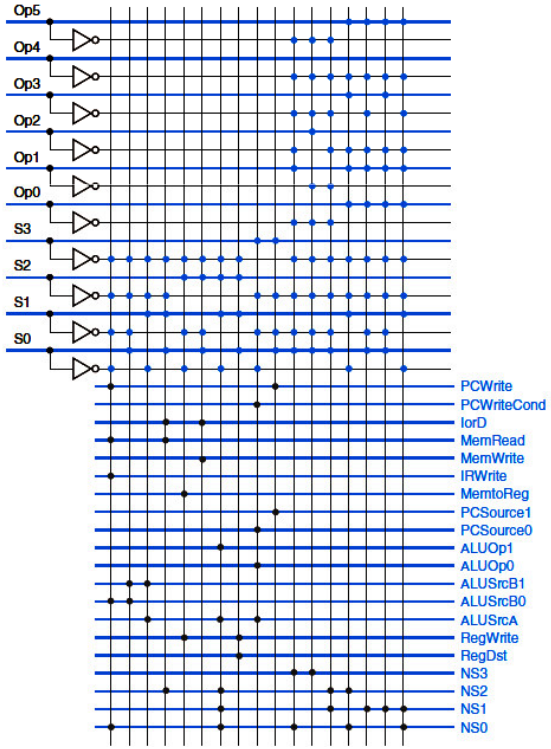
řadič



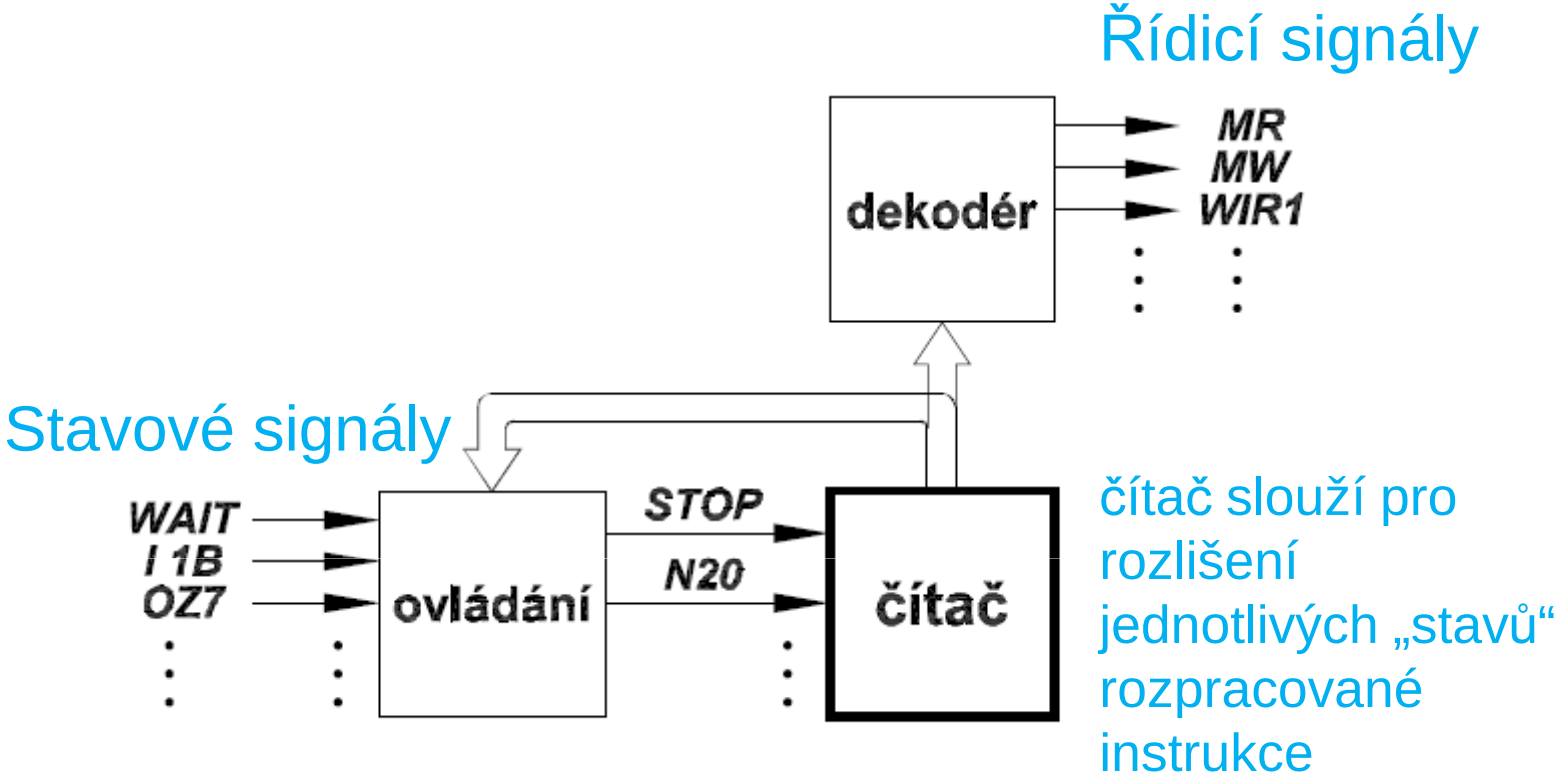
Stavový automat řadiče - příklad

Implementovat „Control logic“ z předchozího slide prakticky můžeme třema způsoby:

- Jako kombinační logický obvod
- Pomocí paměti ROM
  (vstupy do řadiče budou představovat adresní vodiče pro paměť ROM)
- Pomocí PLA (programmable logic array)



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 *řídicí paměť* obsahující *mikroinstrukce*, přičemž každá ze strojových instrukcí je provedena pomocí jedné nebo několika jednodušších mikroinstrukcí.

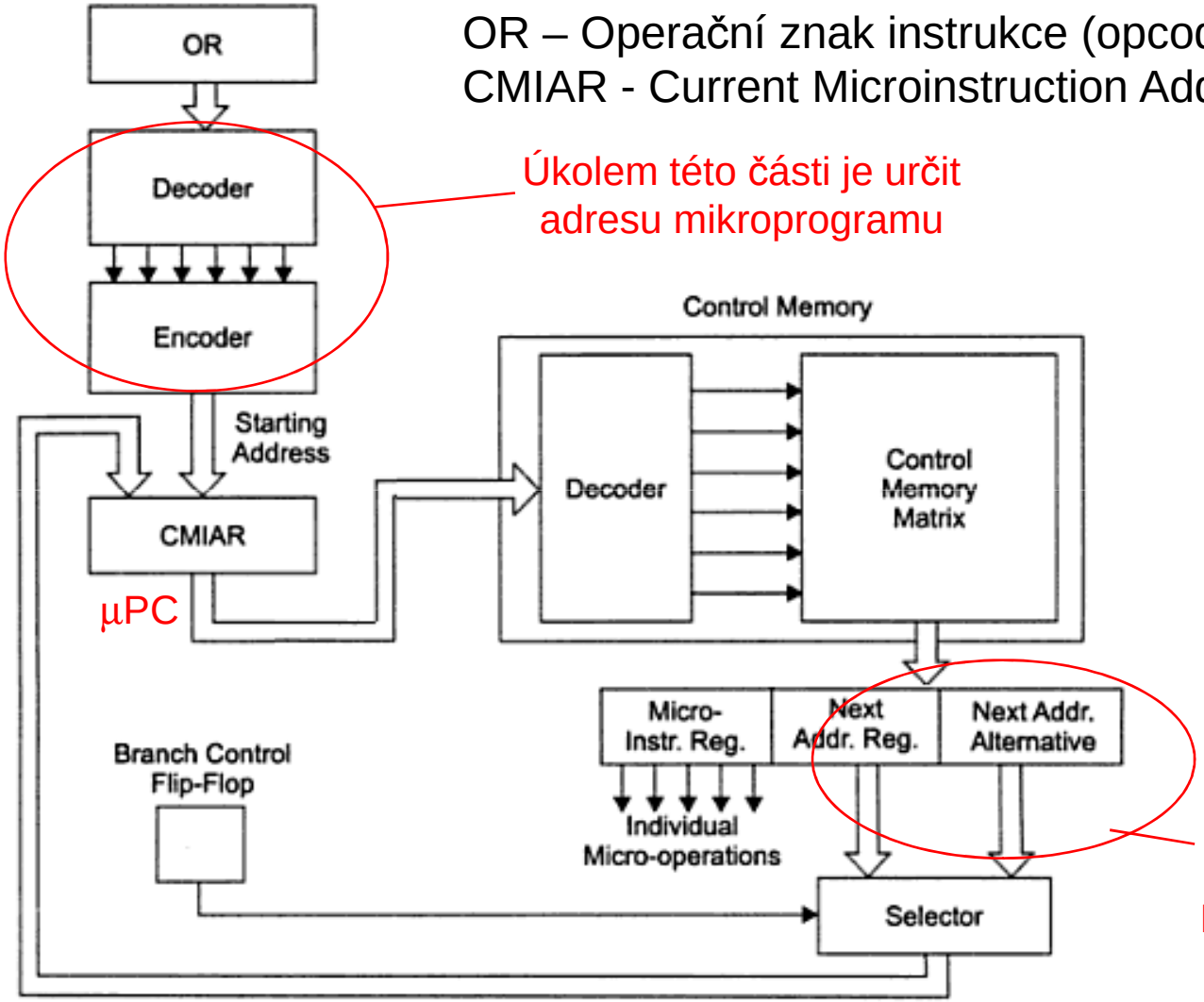
Vstup - kódy instrukcí načtené z operační paměti
Výstup - řídicí signály uvnitř procesorou (ALU, interní sběrnice…), externí signály (paměť,…)

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 (opcode)
CMIAR - Current Microinstruction Address Register (µPC)



Úkolem této části je určit adresu mikroprogramu

Zde může alternativně být tzv. sequencer = Next Address Generator

Mikroprogramovaný řadič

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

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

Jak může vypadat mikroprogram? Realizace instrukce add

Addr.	Mux Ctl	BrUn	BrCON	Brn≠0	Brn=0	End	PC_{out}	MA_{in}	Other Control Signals	Br Addr.	Actions
100	00	0	0	0	0	0	1	1	• • •	XXX	MA ← PC: C ← PC+4;
101	00	0	0	0	0	0	0	0	• • •	XXX	MD ← M[MA]: PC ← C;
102	01	1	0	0	0	0	0	0	• • •	XXX	IR ← MD; μPC ← PLA;
200	00	0	0	0	0	0	0	0	• • •	XXX	A ← R[rb];
201	00	0	0	0	0	0	0	0	• • •	XXX	C ← A + R[rc];
202	11	1	0	0	0	1	0	0	• • •	100	R[ra] ← C; μPC ← 100;

μinstrukce

μoperace

0. PC_{out}, MAR_{in}, Read, Zero A, Set Carry-In, Add, Z_{in}
1. Z_{out}, PC_{in}, Wait MFC
2. MDR_{out}, IR_{in}
3. $AddressFieldOfIR_{out}$, MAR_{in}, Read
4. $R1_{out}$, Y_{in}, Wait MFC
5. MDR_{out}, Add, Z_{in}, Set CC
6. Z_{out}, $R1_{in}$, End

John Franco: What is Microprogramming and Why Should We Know About it?
http://gauss.ececs.uc.edu/Courses/c4029/exams/Spring2013/Review/microcode.pdf

Mikroprogramovaný řadič

Organizace mikroinstrukcí – horizontální, vertikální a diagonální

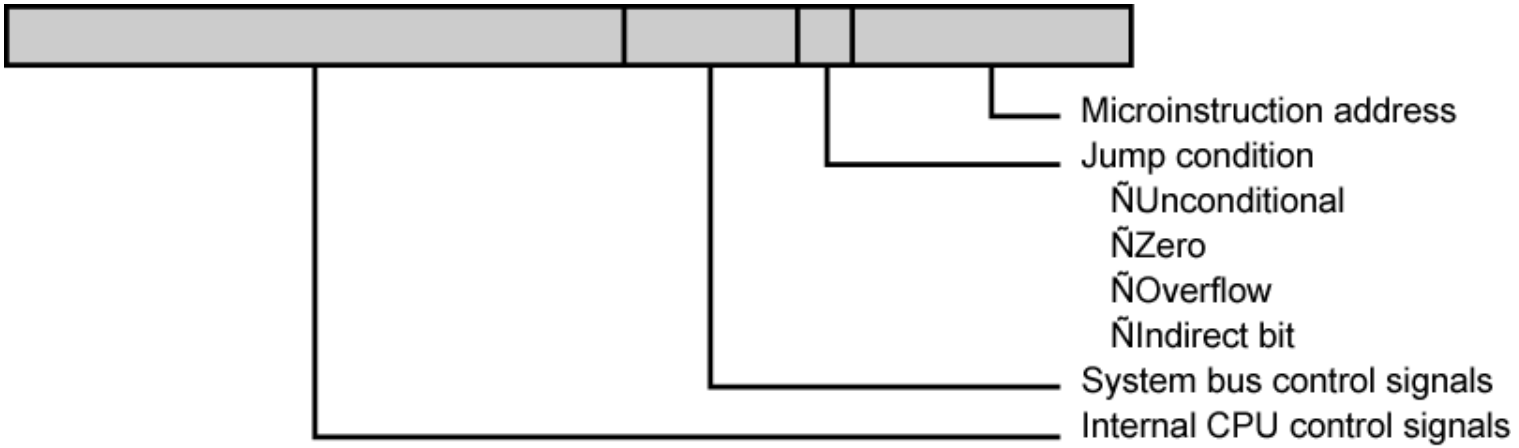
Vertikální:

- 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.

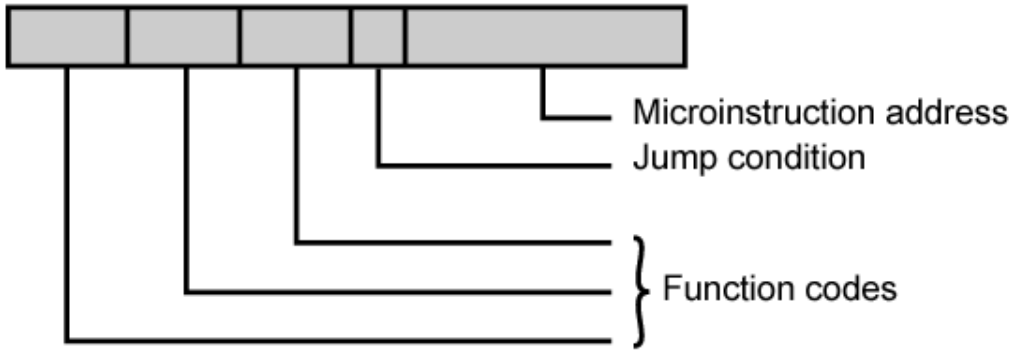
Horizontální:

- 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

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).

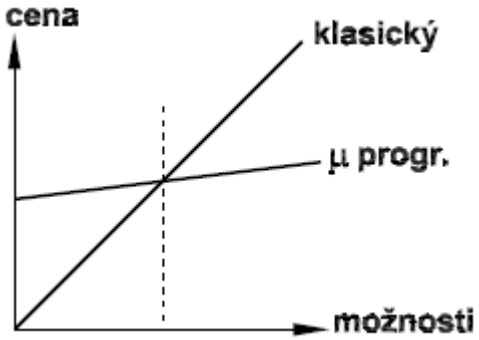
Mikroprogramovaný řadič je vlastně počítačem v počítači.

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.

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.
- 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.