Architektura počítačů
Implementace procesoru

http://d3s.mff.cuni.cz/teaching/computer_architecture/

Lubomír Bulej

bulej@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Ukázková architektura: MIPS

- **Základní charakteristika**
  - Zjednodušená oproti reálným implementacím
    - Z důvodu názornosti, ale stále dostatečně silná
  - Registry
    - 32 obecných 32-bitových registrů R0 – R31 (*general purpose*)
    - Registr PC s adresou následující instrukce pro dekódování
    - Speciální řídící registry
      - Adresa instrukce, která vyvolala výjimku, apod.

Ukázková architektura: MIPS (2)

- **Paměť**
  - Přístup pouze na zarovnané adresy (dělitelné 4)
    - Odpovídá délce slova 32 bitů
  - Nepřímá adresace s posunutím *(indirect with immediate displacement)*
    - **Load:** R2 := mem[R1 + immediate]
    - **Store:** mem[R1 + immediate] := R2

Ukázková architektura: MIPS (3)

- **Operace**
  - Tříoperandové operace
    - Zdrojové operandy: registr/registr, registr/immediate
    - Cílový operand: registr
    - Aritmetické a logické operace, přesun dat mezi registry
  - Architektura load/store
    - Přesuny dat mezi pamětí a registry
  - Podmíněné skoky
    - Při rovnosti/nerovnosti obsahu dvou registrů
  - Nepodmíněné skoky
    - Včetně nepřímých skoků, volání a návrat z podprogramu
  - Speciální instrukce

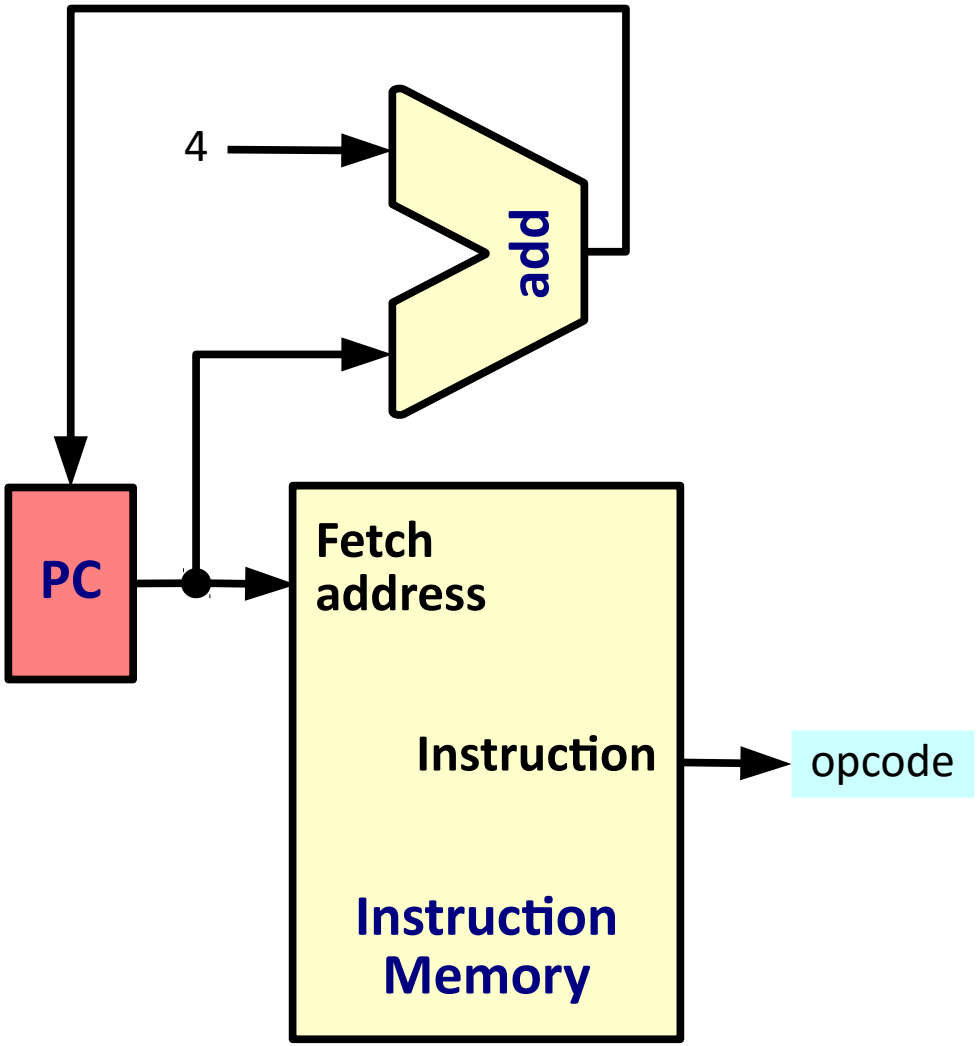
Jednocyklová datová cesta

- **Základní uspořádání logických celků procesoru**
  - Spojení kombinačních a sekvenčních obvodů
  - Veškeré operace provedeny v jednom cyklu
    - Zpracování každé instrukce má jedinou fázi (jeden atomický krok)
    - Vhodné pro operace přibližně srovnatelné složitosti
    - Časový signál považujeme za implicitní
  - **Zjednodušení:** Oddělená instrukční paměť (Harvardská architektura)

Ukázková architektura: MIPS (3)

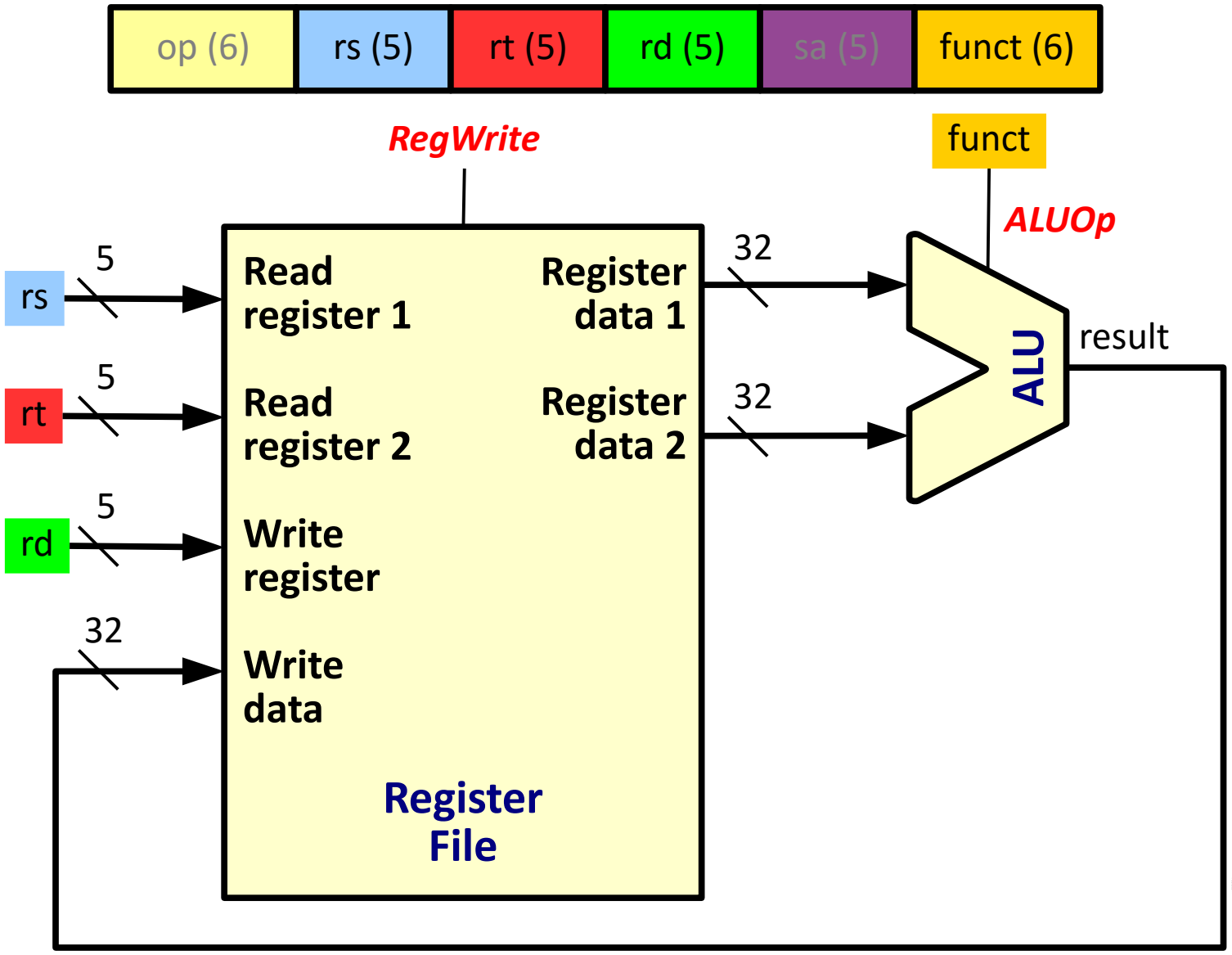
- **Kroky zpracování instrukce**
  1. Čtení kódu instrukce z paměti na adrese PC
  2. Dekódování instrukce a čtení operandů
  3. Vykonání operace odpovídající instrukčnímu kódu
     - Operace s obsahem registrů, výpočet adresy pro čtení (zápis) v paměti, porovnávání operandů pro podmíněný skok
  4. Uložení výsledku operace
     - Uložení výsledku do registru, čtení (zápis) v paměti
  5. Posun PC na následující instrukci
     - Bezprostředně následující, vyjma podmíněného/nepodmíněného skoku nebo výjimky

Sekvenční obvod čtení instrukce (fetch)

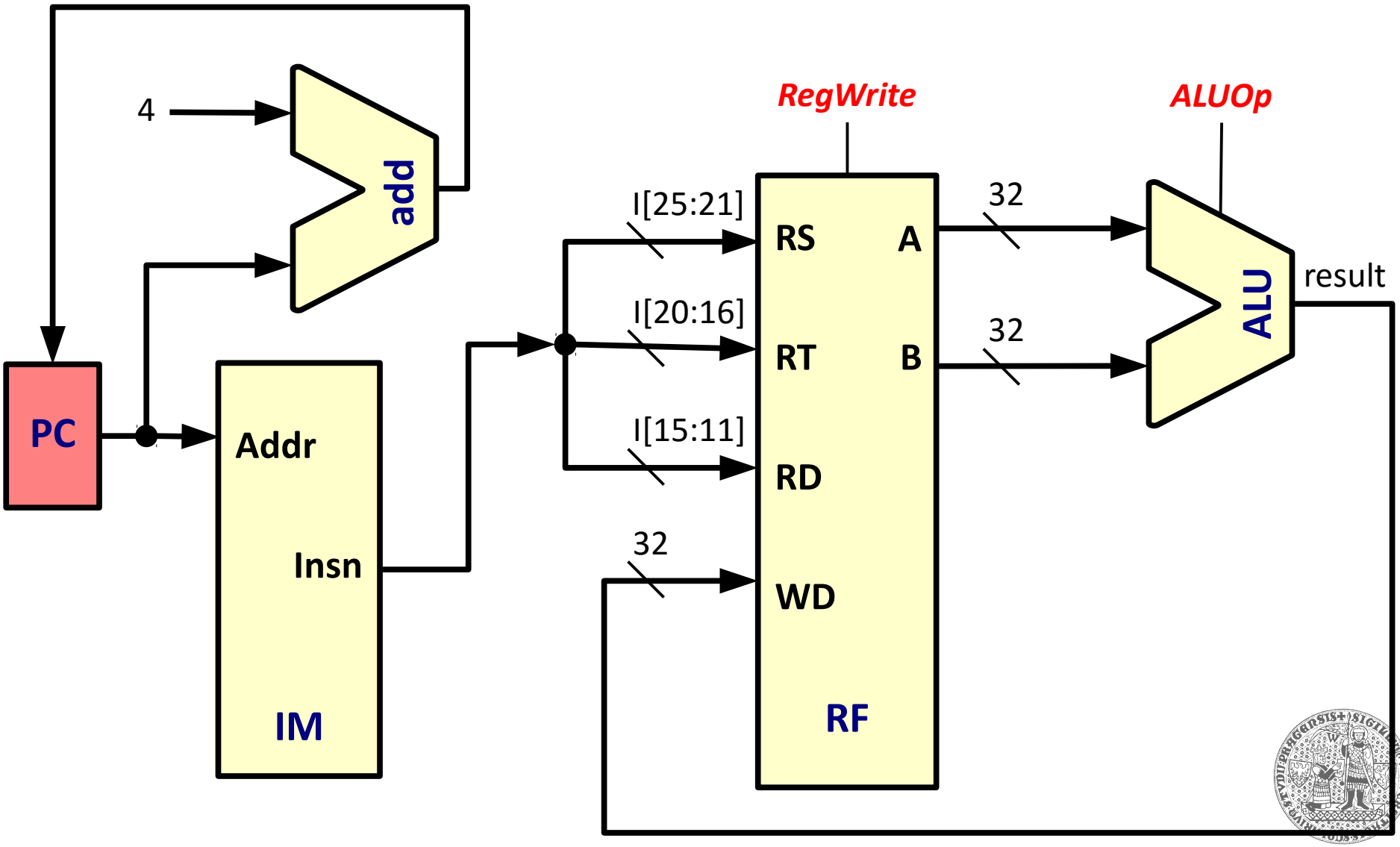


- **Registr PC**
  - Adresa instrukce v paměti
  - Pro programátora není přímo přístupný
- **Sčítačka**
  - Inkrementace PC o 4
  - Implicitní posun na následující instrukci

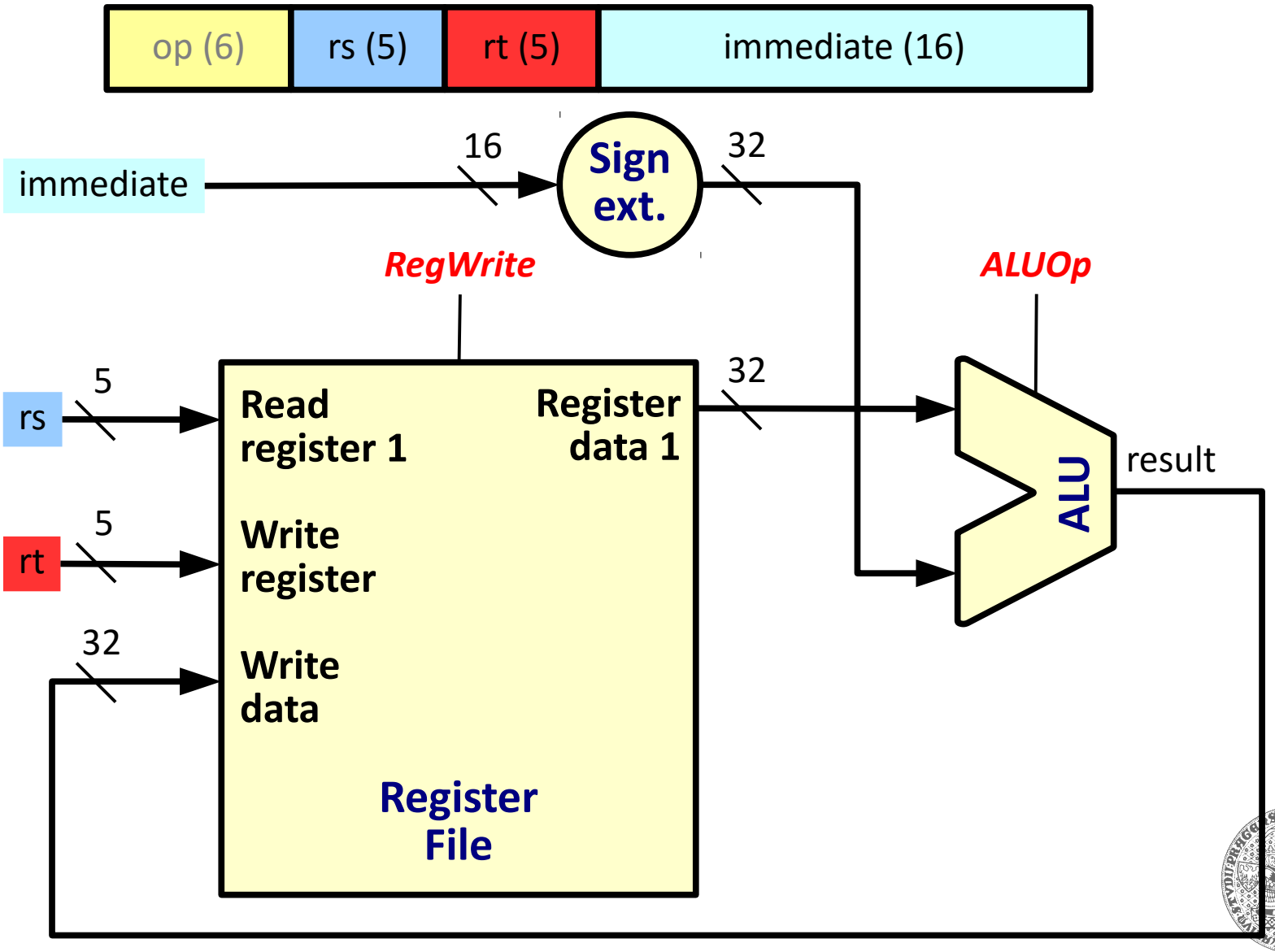
Registrové operace (add, sub, ...)



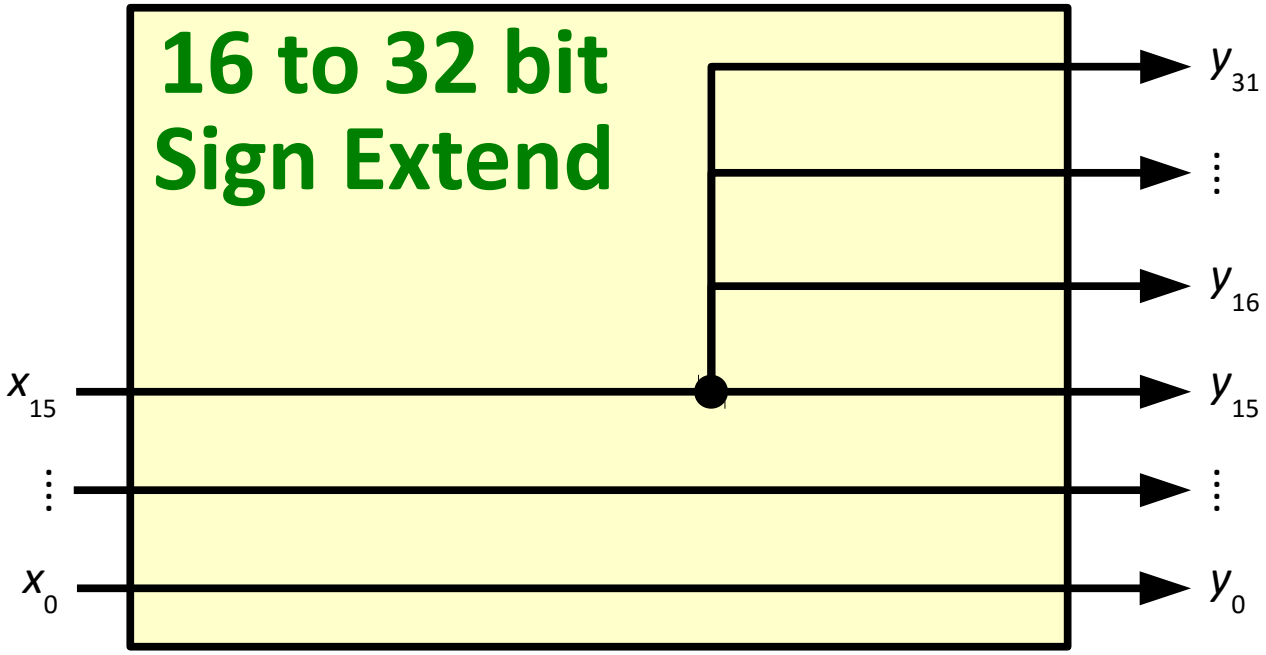
Podpora registrových operací



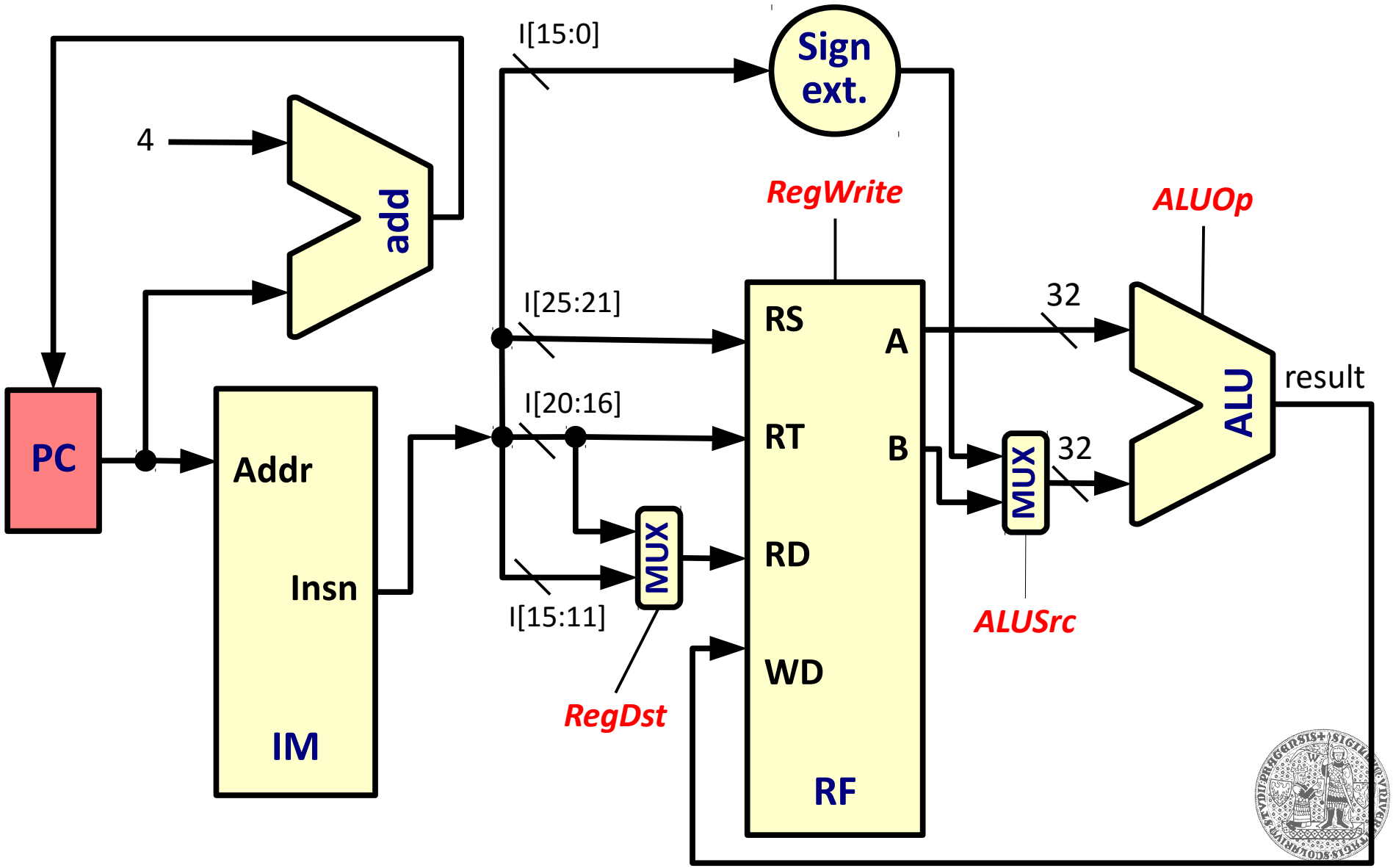
Operace s přímými operandy (addi, ...)



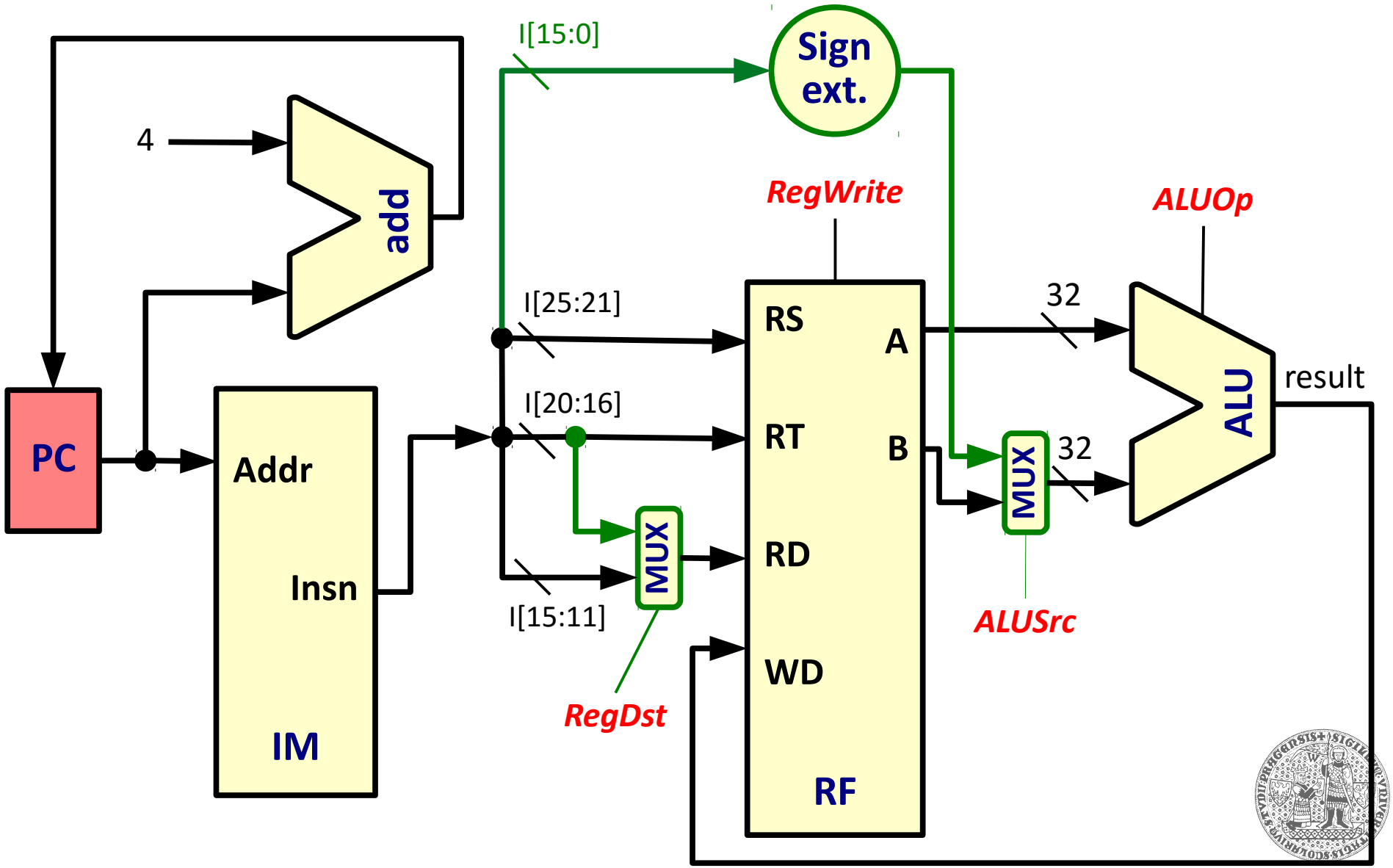
Logický obvod znaménkového rozšíření



Podpora přímých operandů

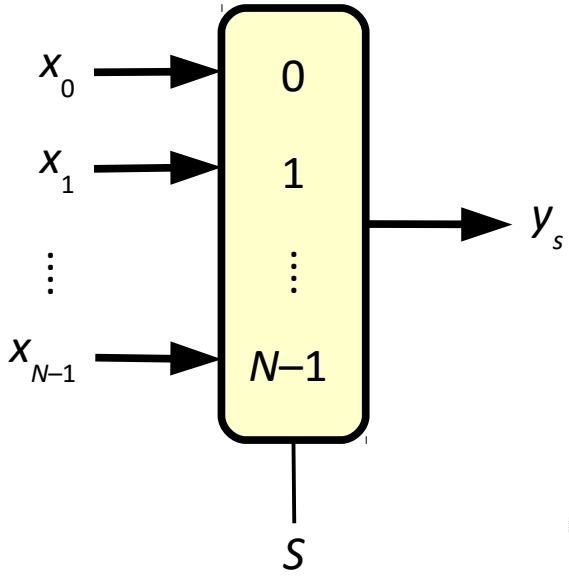


Podpora přímých operandů



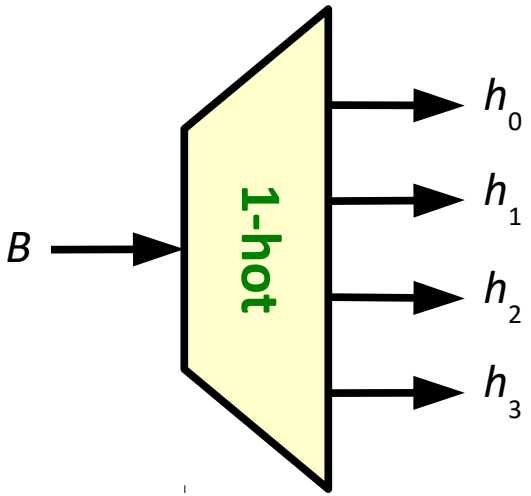
Multiplexer (mux)

- **Přepínač vstupů**
  - Logický prvek pro výběr vstupu
    - **Selektor:** n-bitové číslo $S \in \{0, \ldots, 2^{n-1}\}$
    - **Vstup:** $N=2^n$ m-bitových hodnot $x_0, x_1, \ldots, x_{N-1}$
    - **Výstup:** m-bitová hodnota $y=x_S$



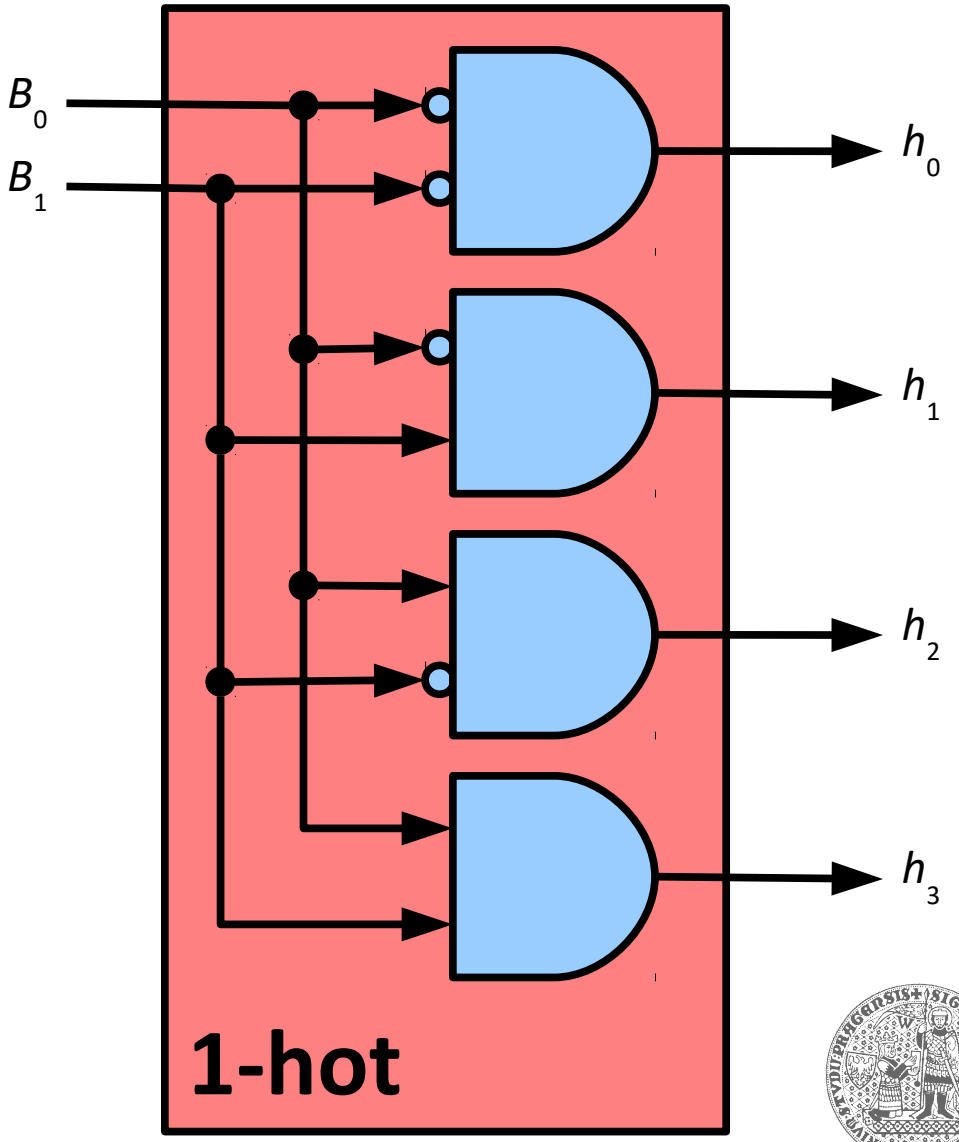
Binární dekodér do kódu „1 z *N*“

- ***Binary to 1-hot***
  - Logický prvek pro aktivaci 1 z *N* výstupů na základě hodnoty na vstupu
    - **Vstup:** *n*-bitové číslo $B \in \{0, \ldots, 2^{n-1}\}$
    - ***N*=2^n výstupů:** *B*-tý výstup logická 1 (hot), ostatní logická 0

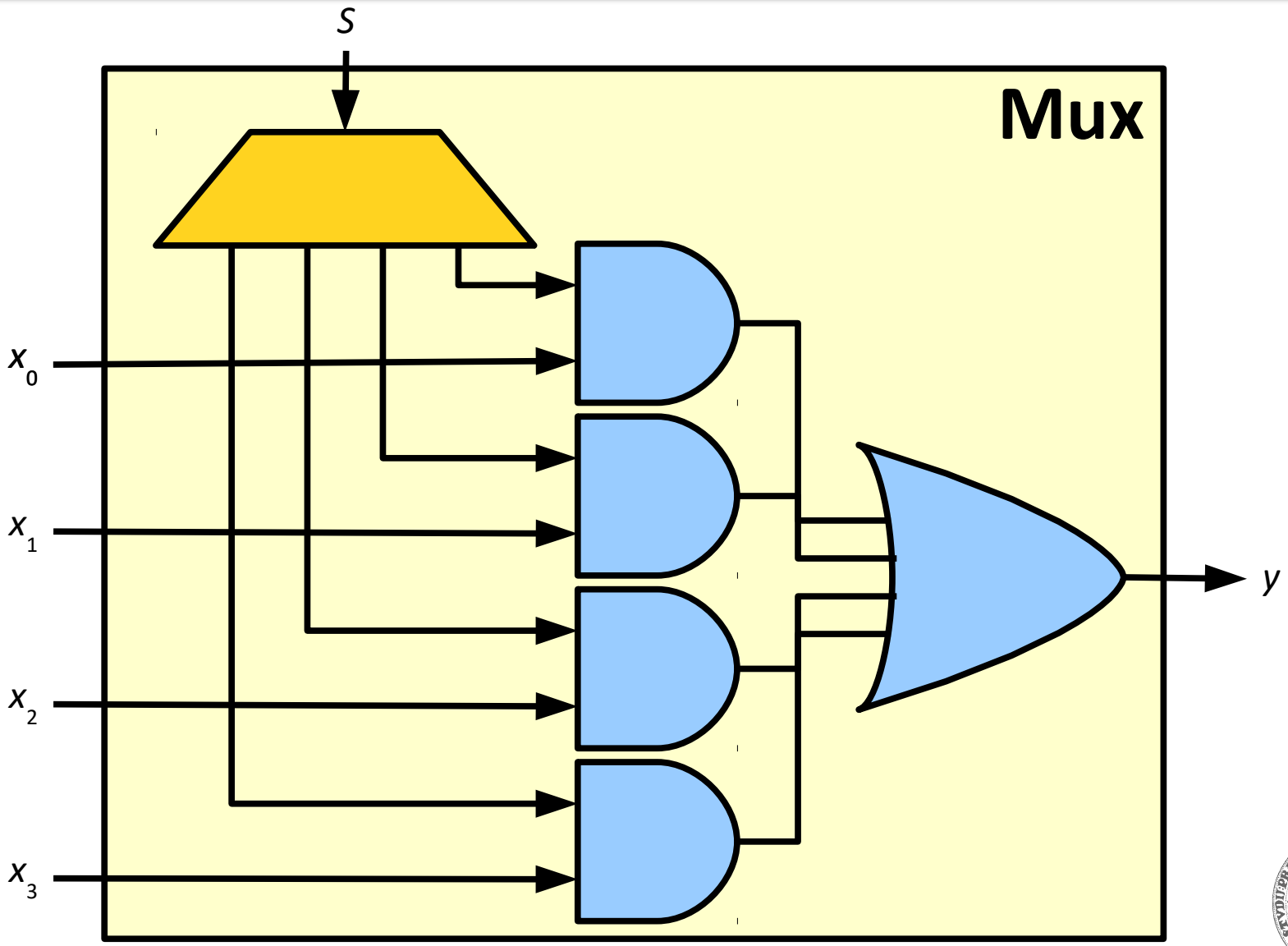


Binární dekodér pro N=4

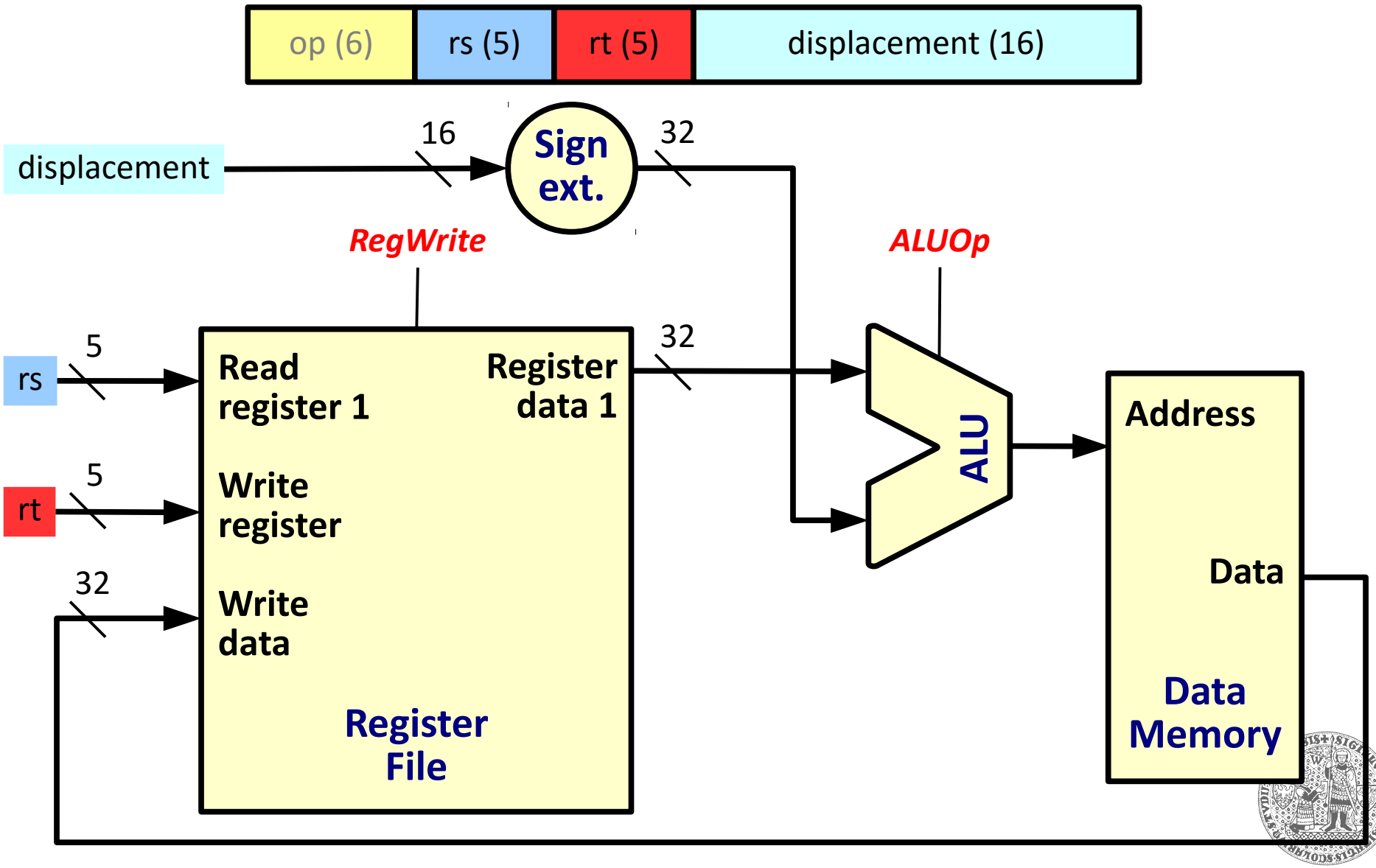
Vstupy		Výstupy			
B_1	B_0	h_3	h_2	h_1	h_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



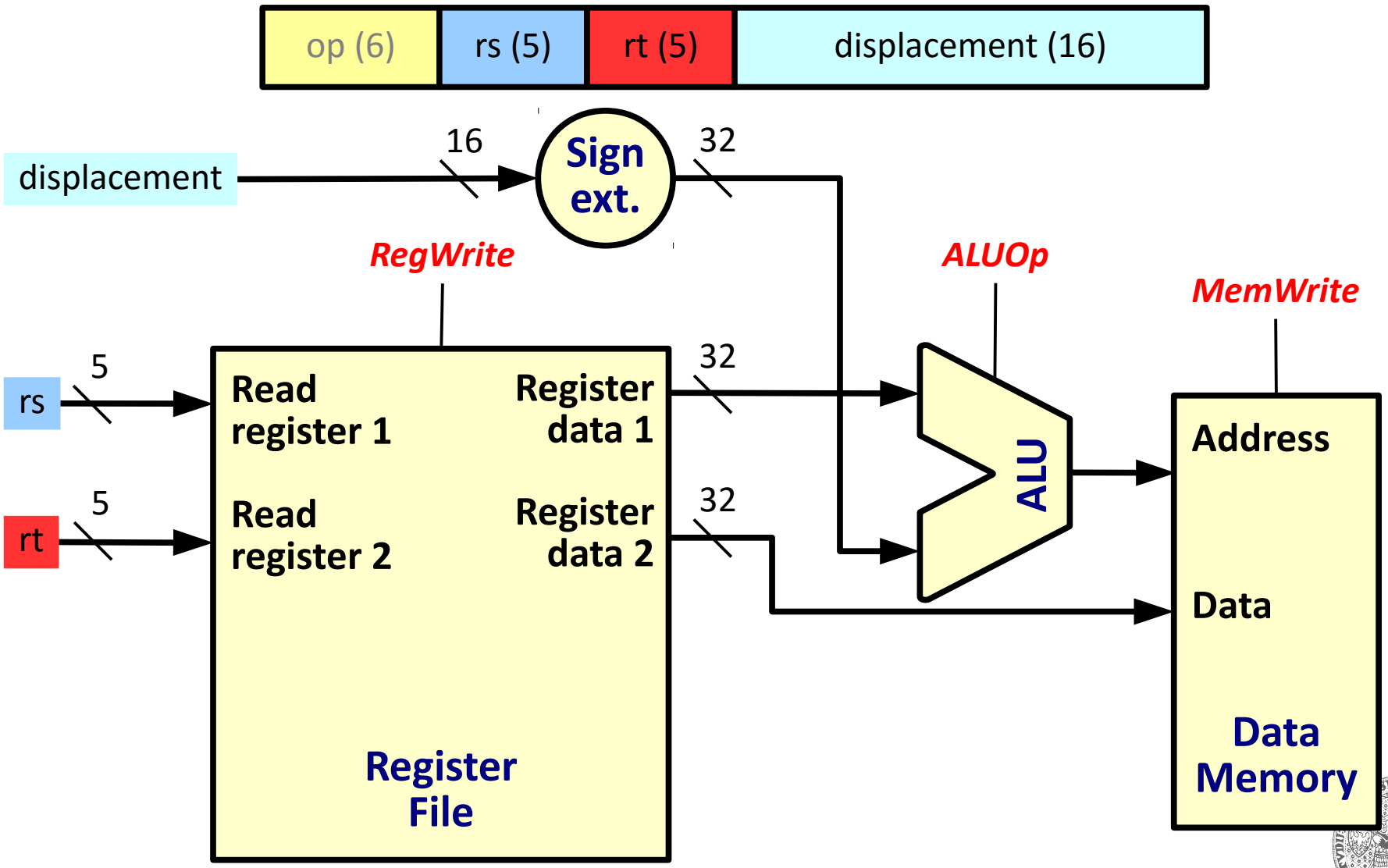
Přepínač vstupů pro N=4, m=1



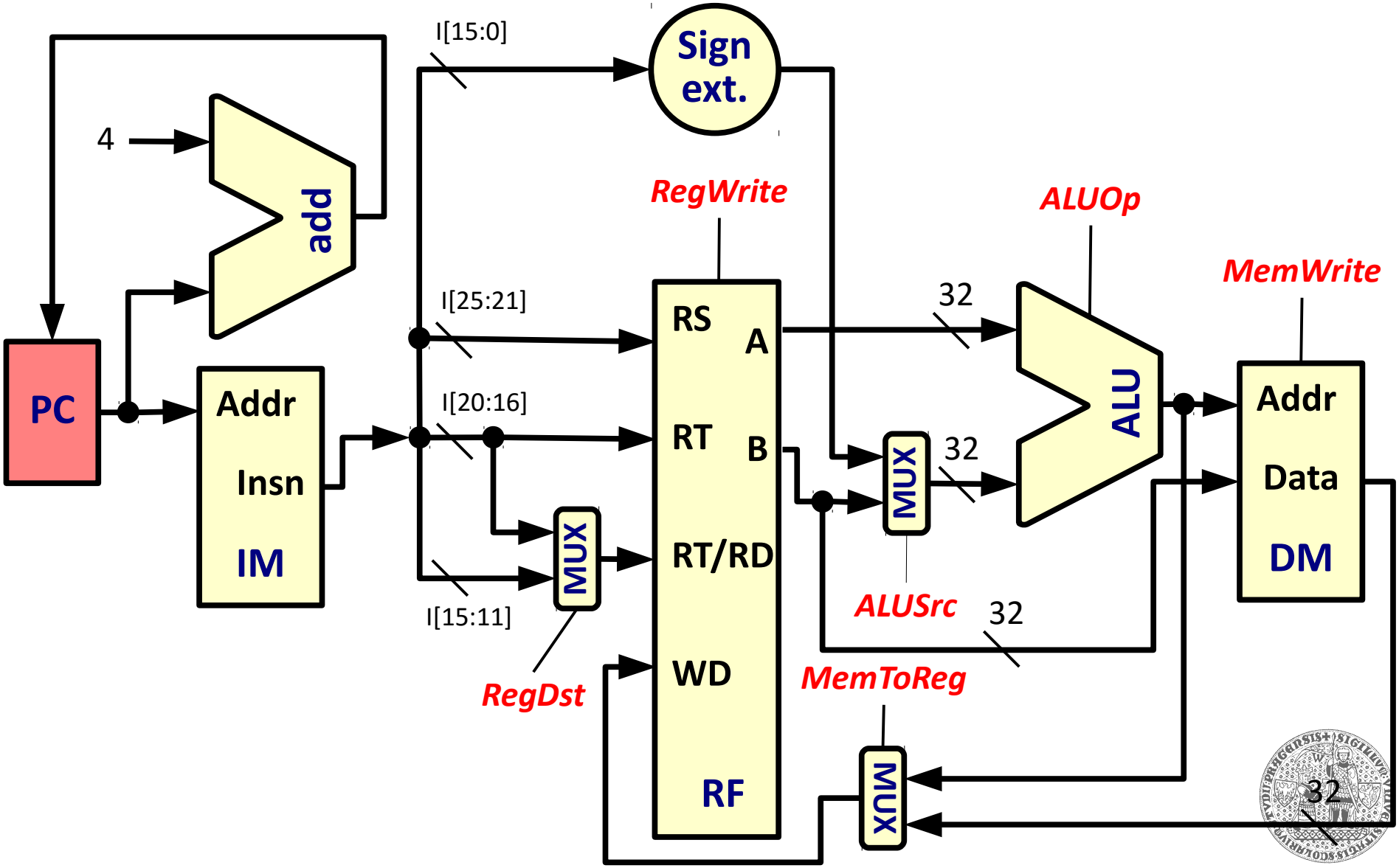
Čtení dat z paměti (ld)



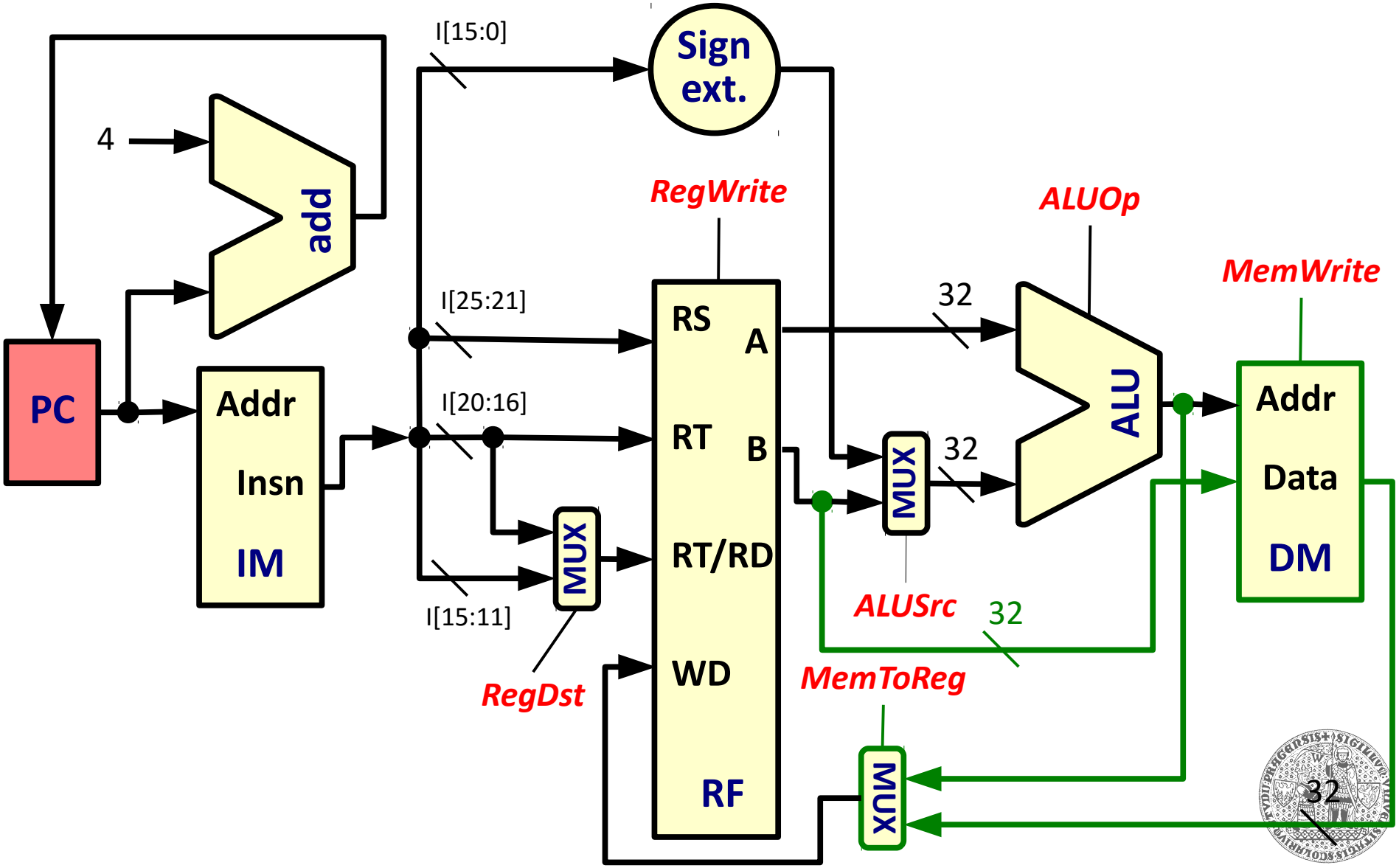
Zápis dat do paměti (st)



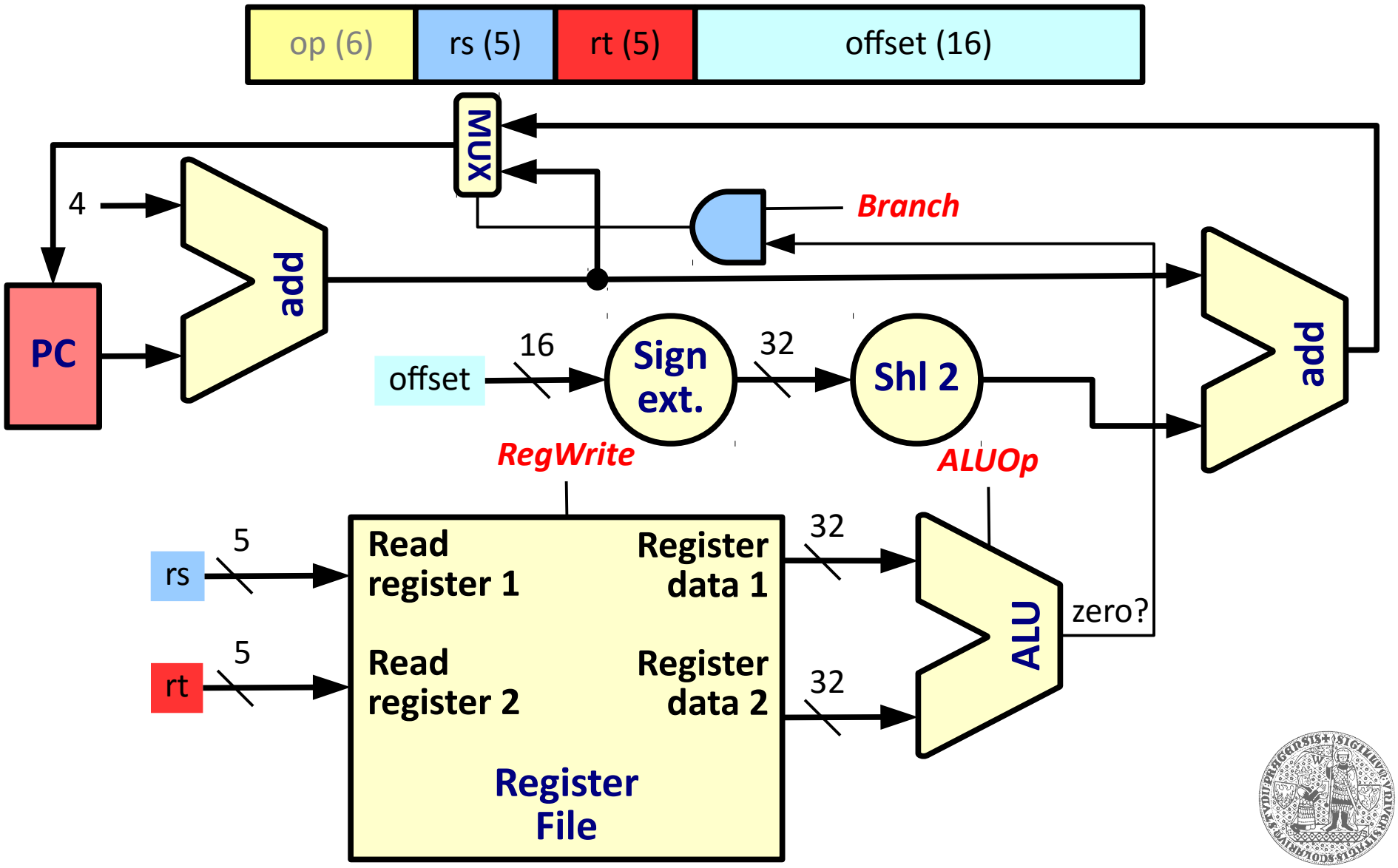
Podpora přístupu do paměti



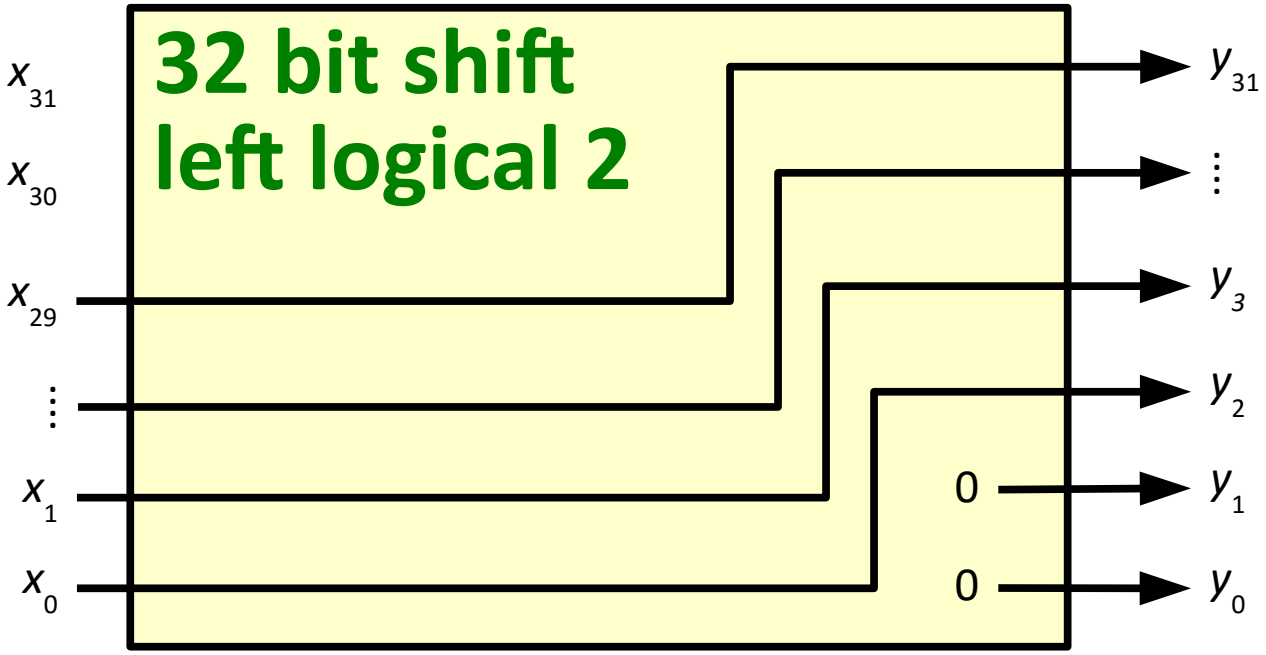
Podpora přístupu do paměti



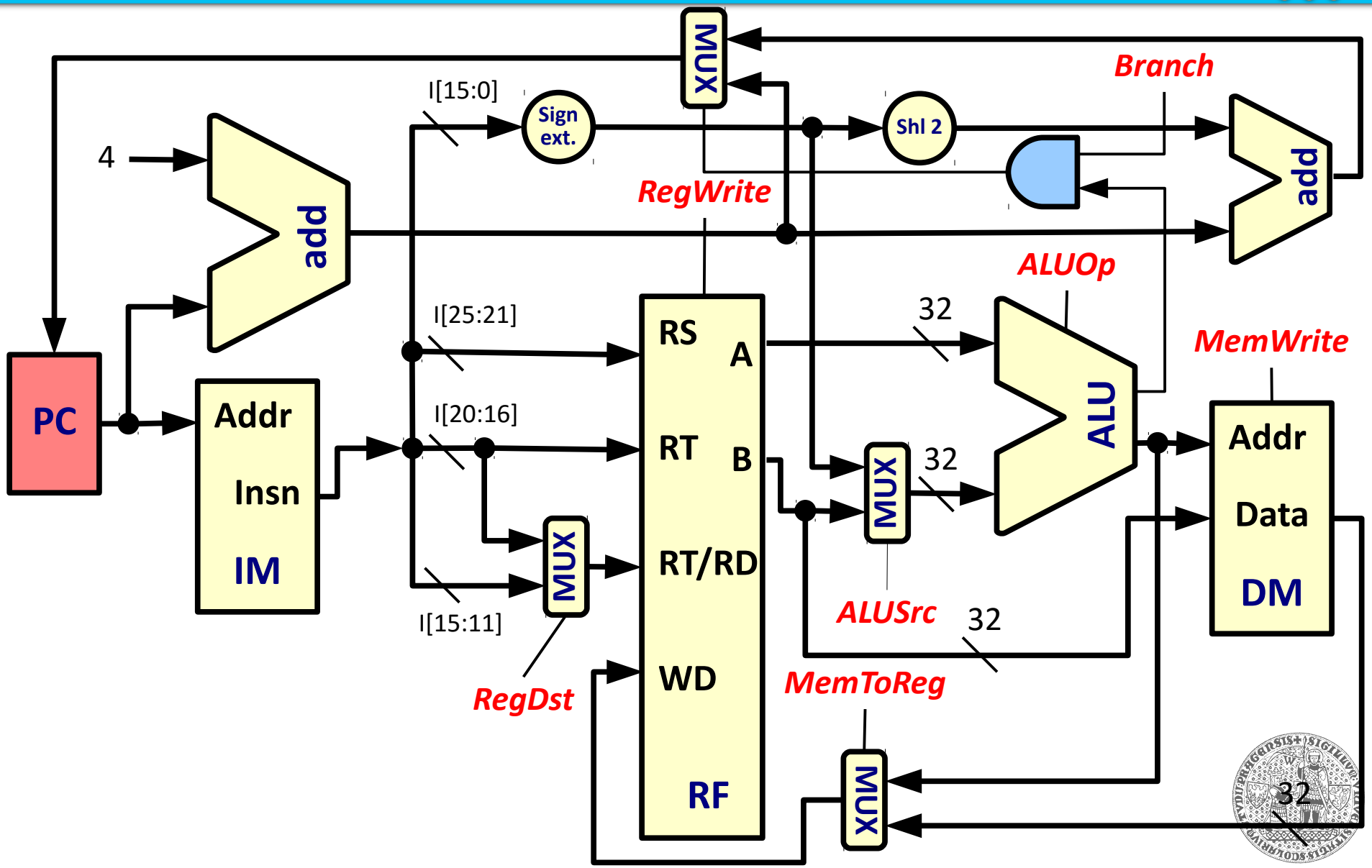
Podmíněný skok s relativní adresou (b)



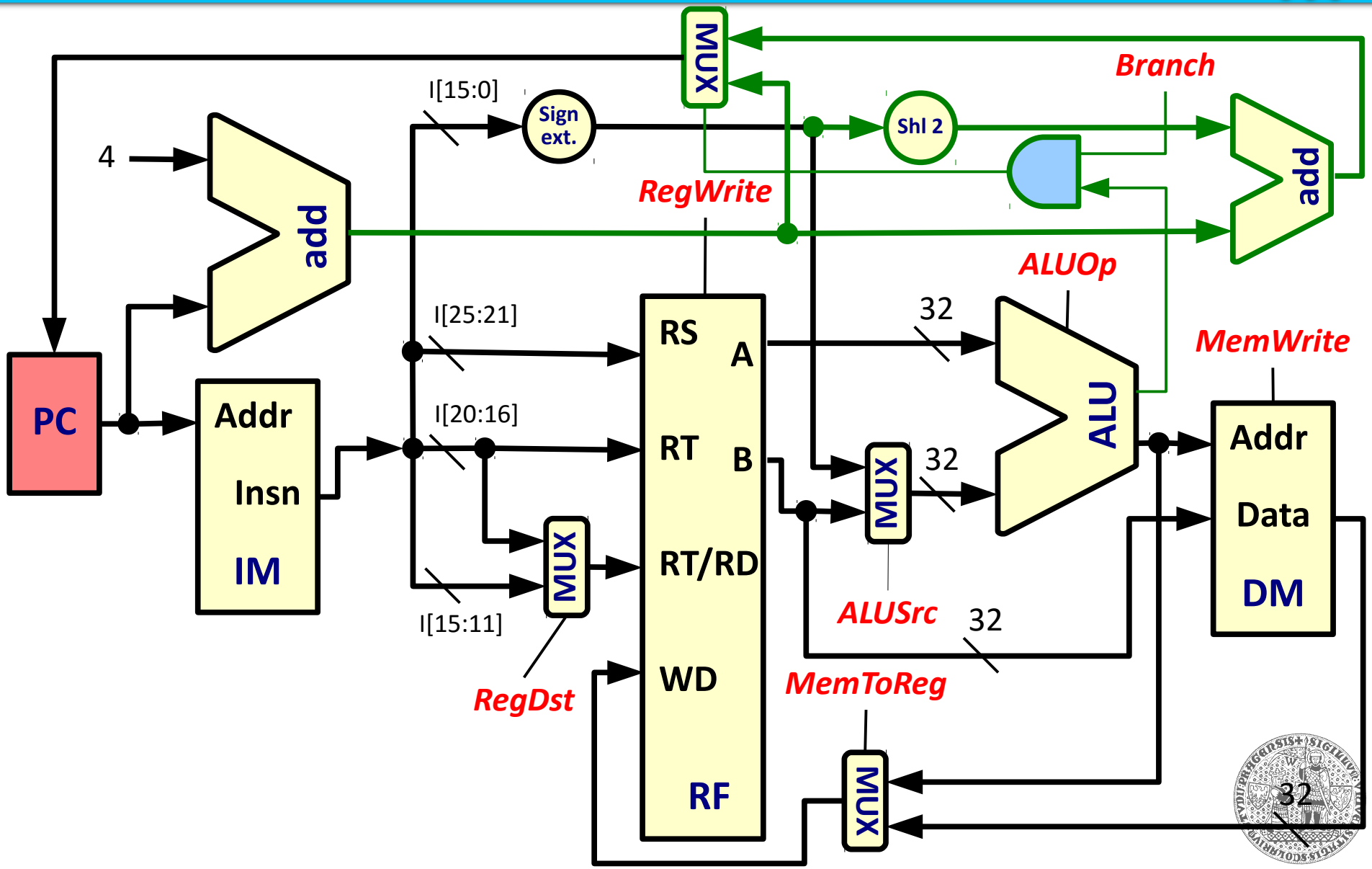
Logický obvod logického posunu vlevo



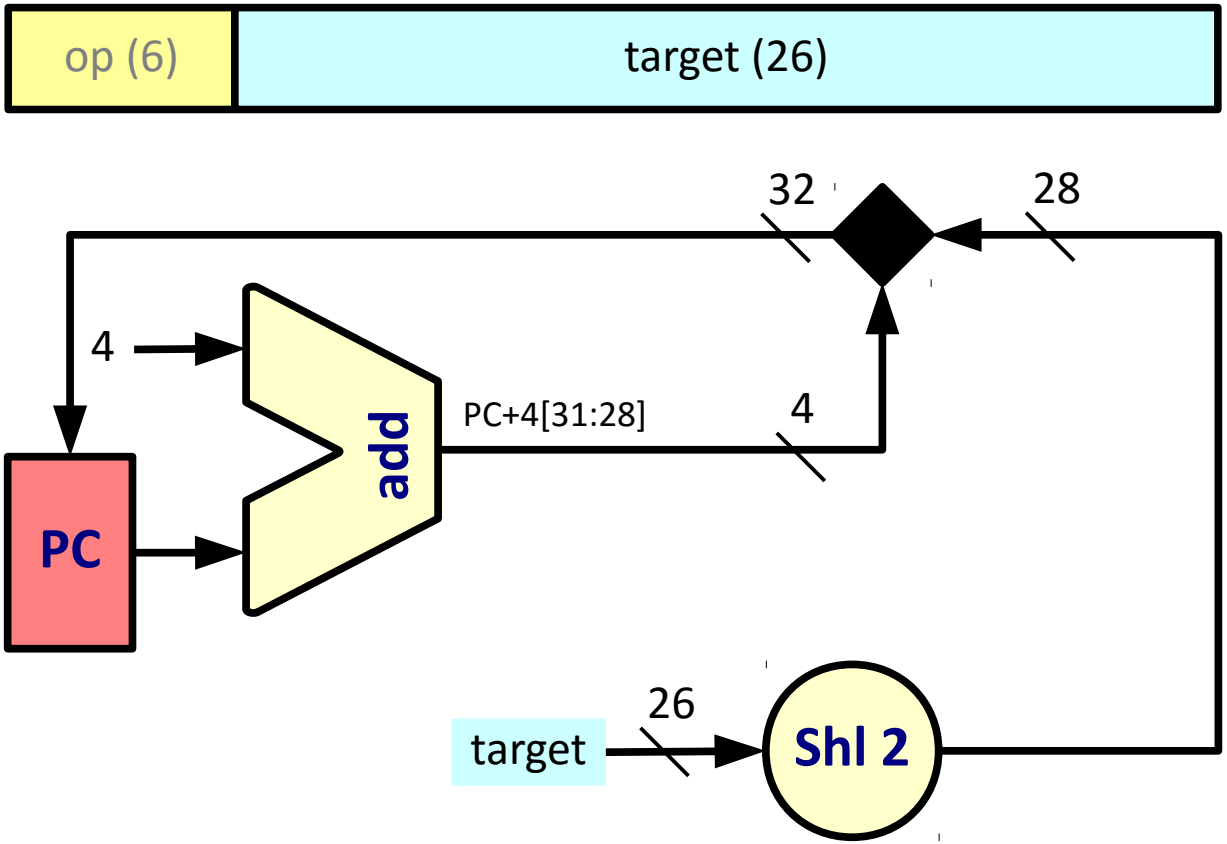
Podpora podmíněného skoku



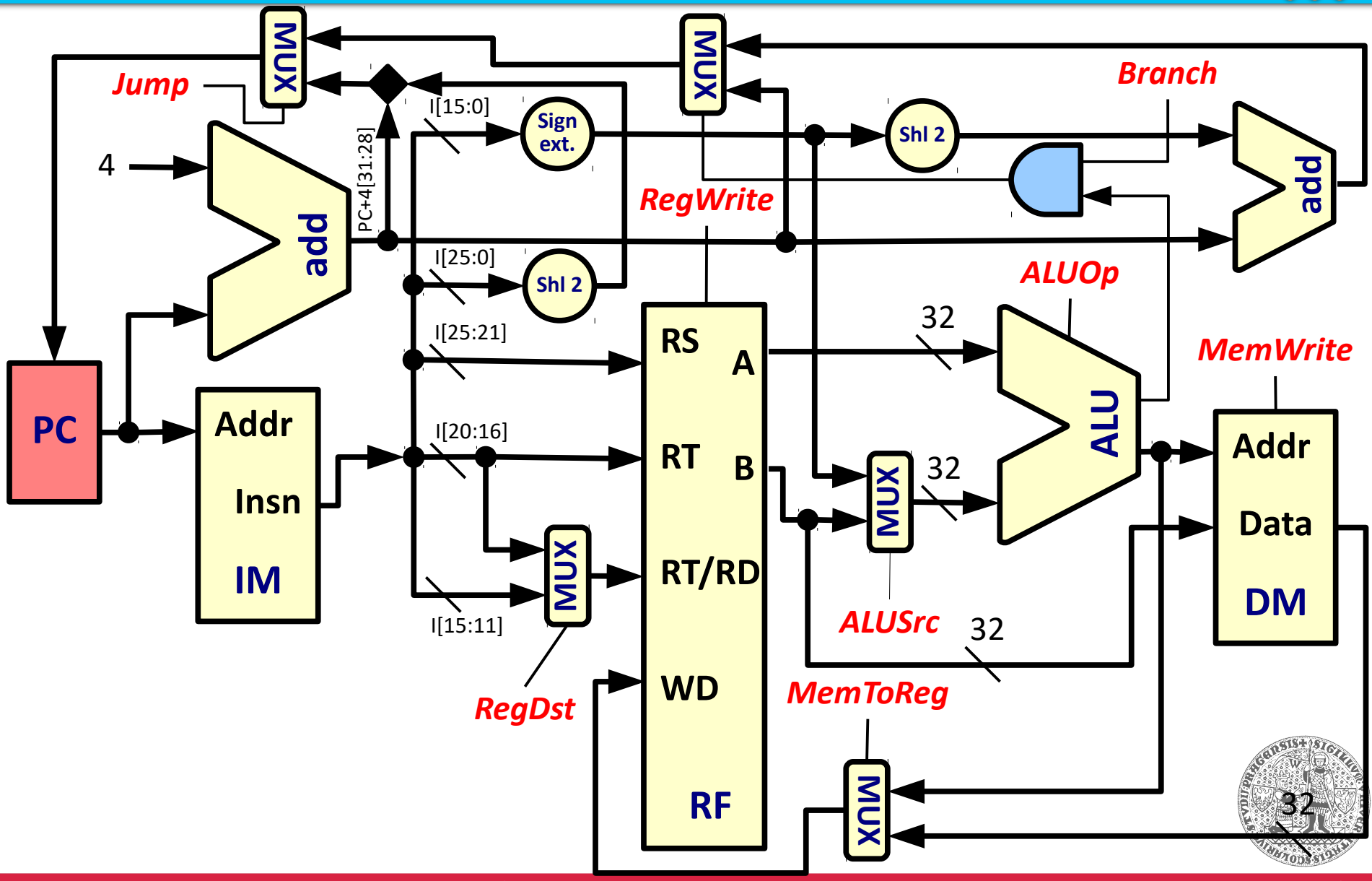
Podpora podmíněného skoku



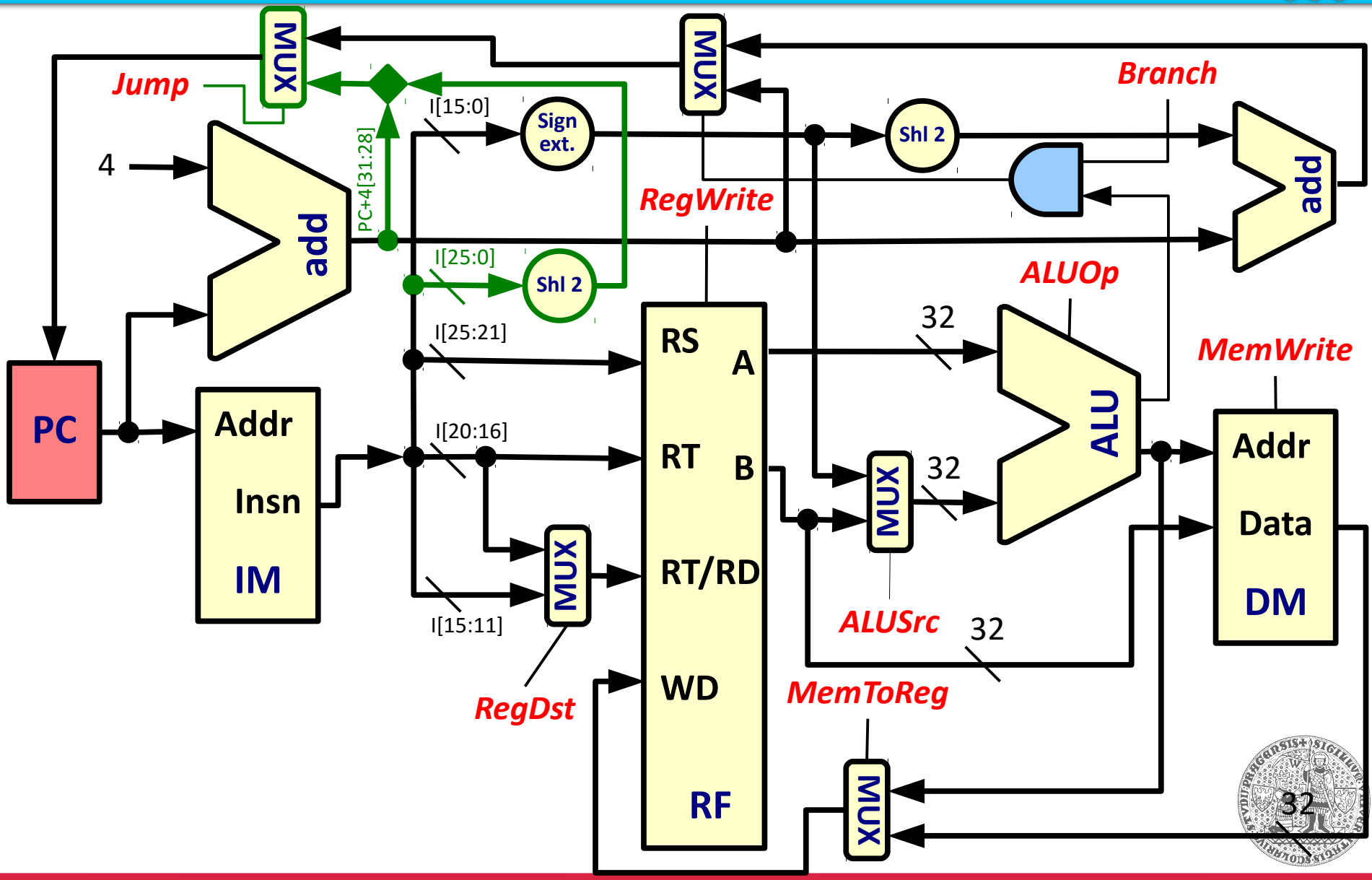
Nepodmíněný absolutní skok (j)



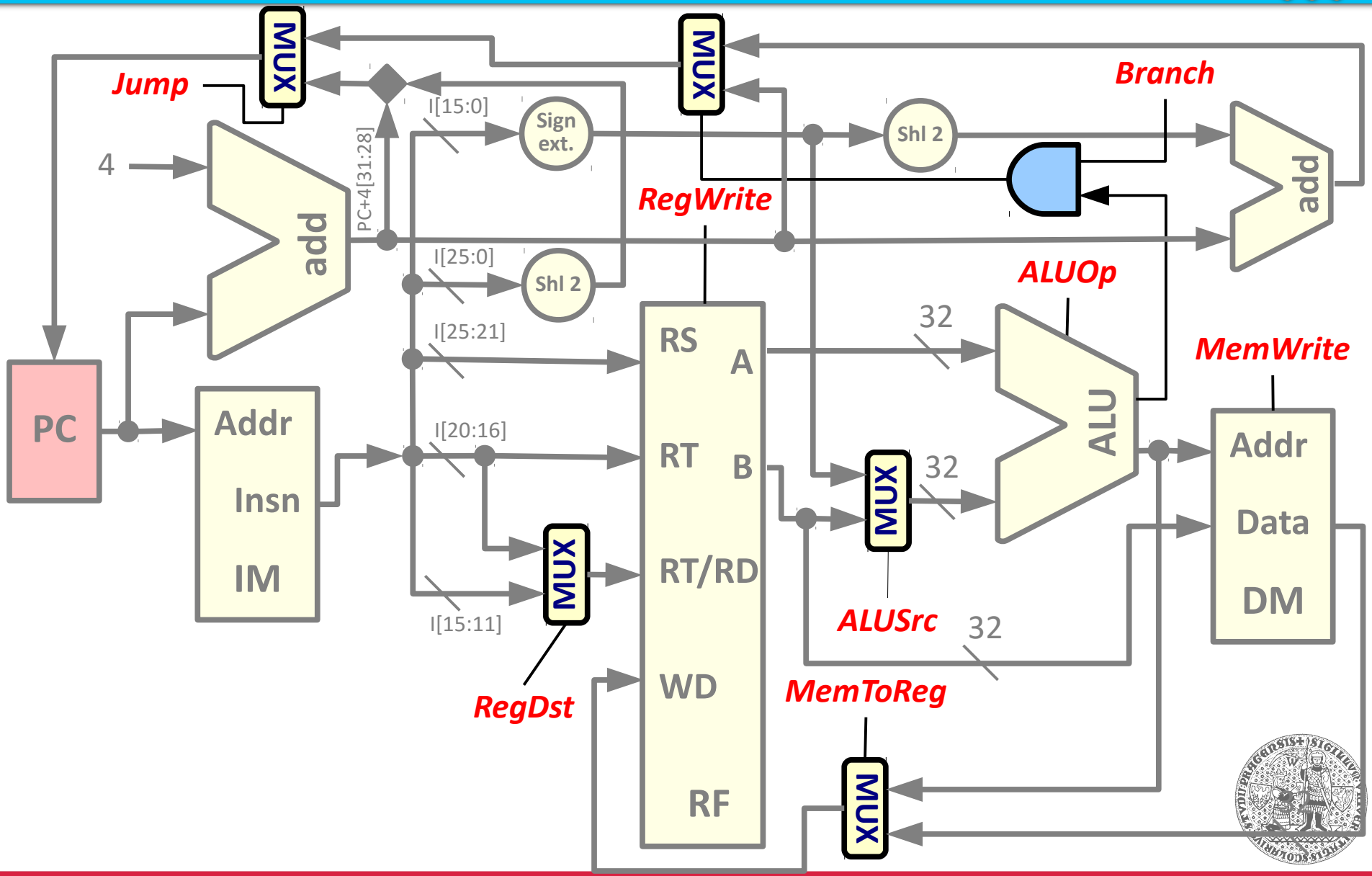
Podpora nepodmíněného skoku



Podpora nepodmíněného skoku



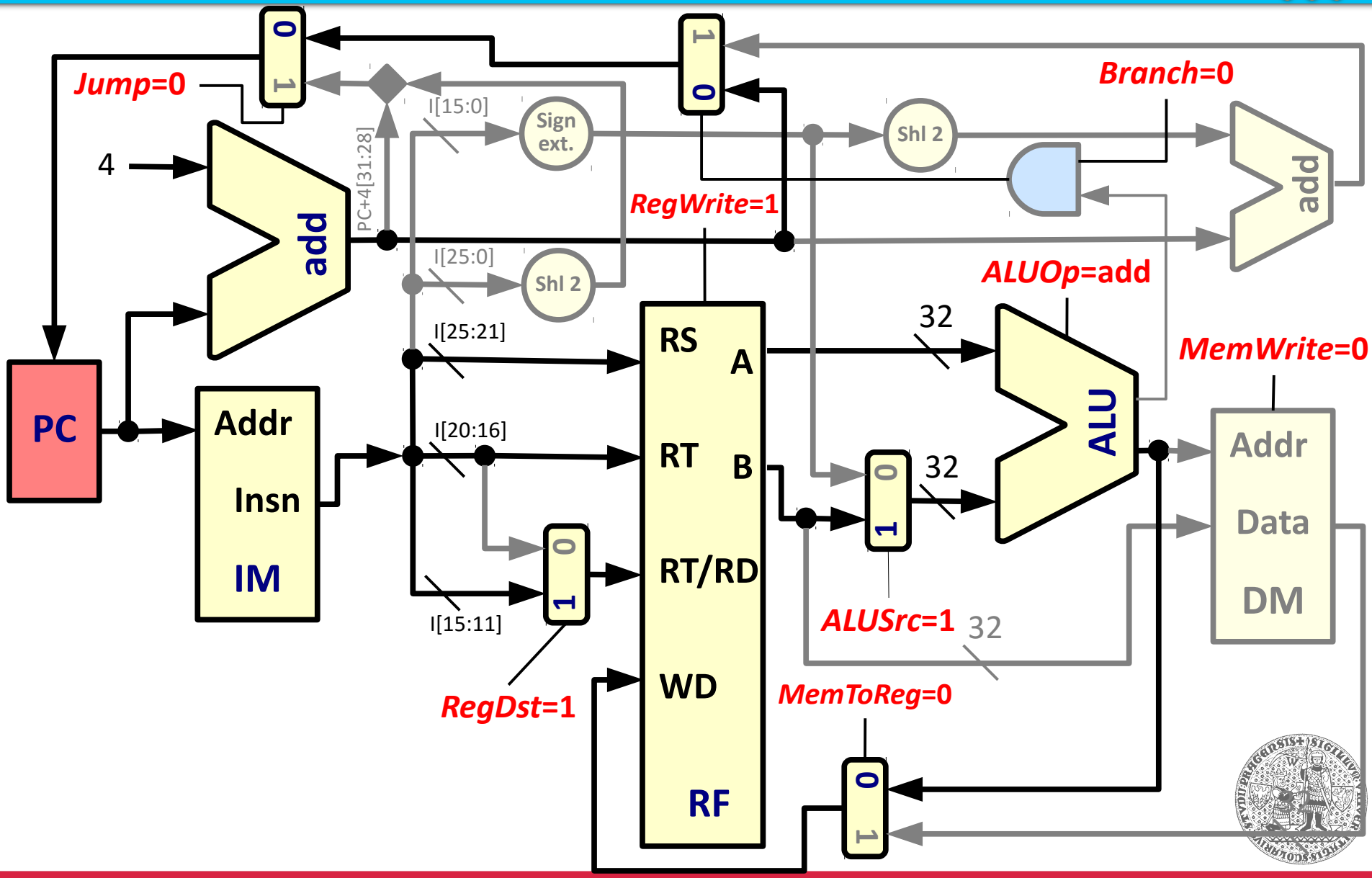
Řízení jednocyklové datové cesty



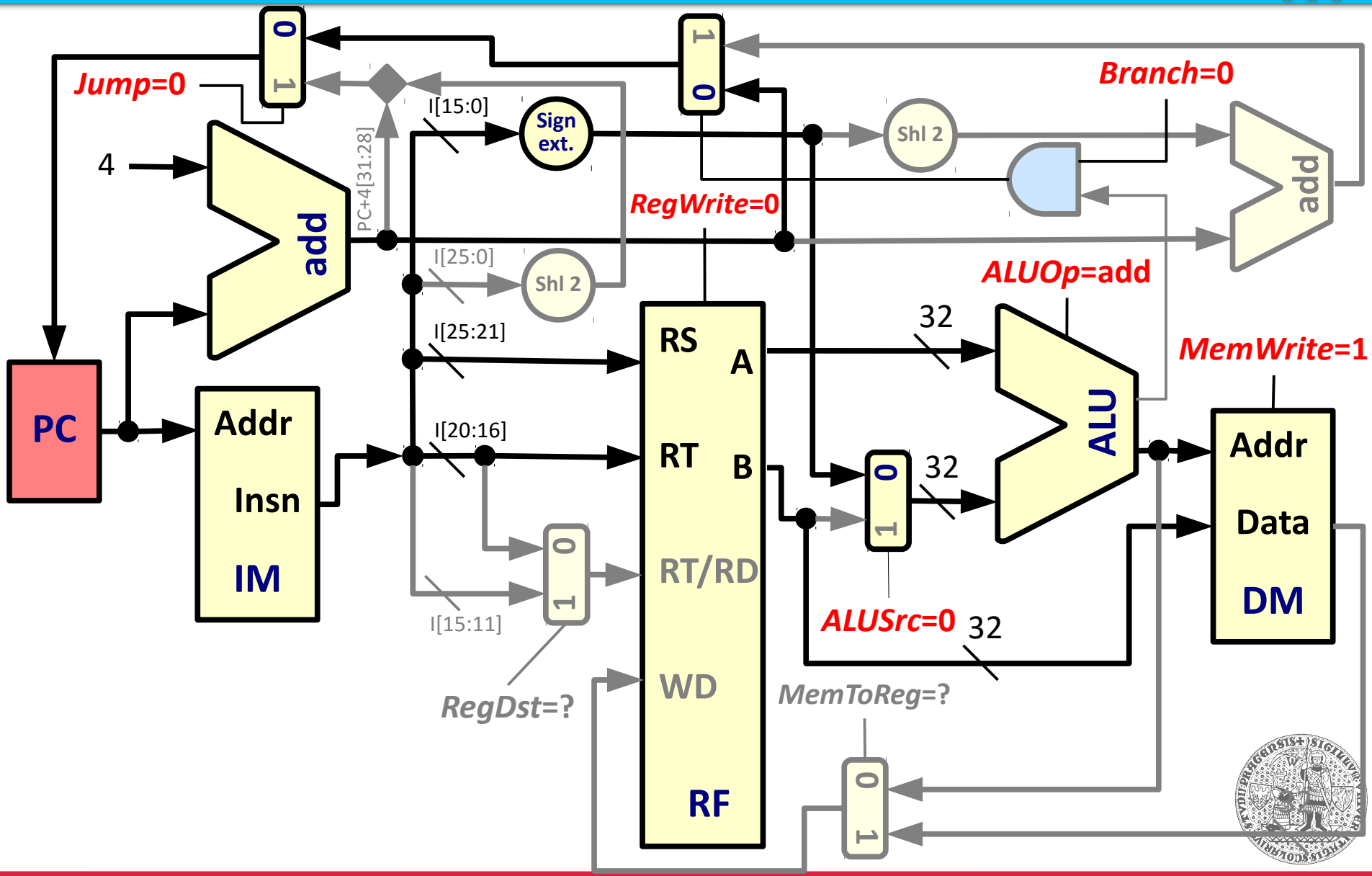
Řízení jednocyklové datové cesty (2)

- **Řízení průchodu dat**
  - V závislosti na typu operace
  - Generování řídících signálů
    - Zdroj hodnoty PC
    - Zápis do registrů
    - Čtení/zápis paměti
    - Operace ALU
    - Nastavení multiplexerů

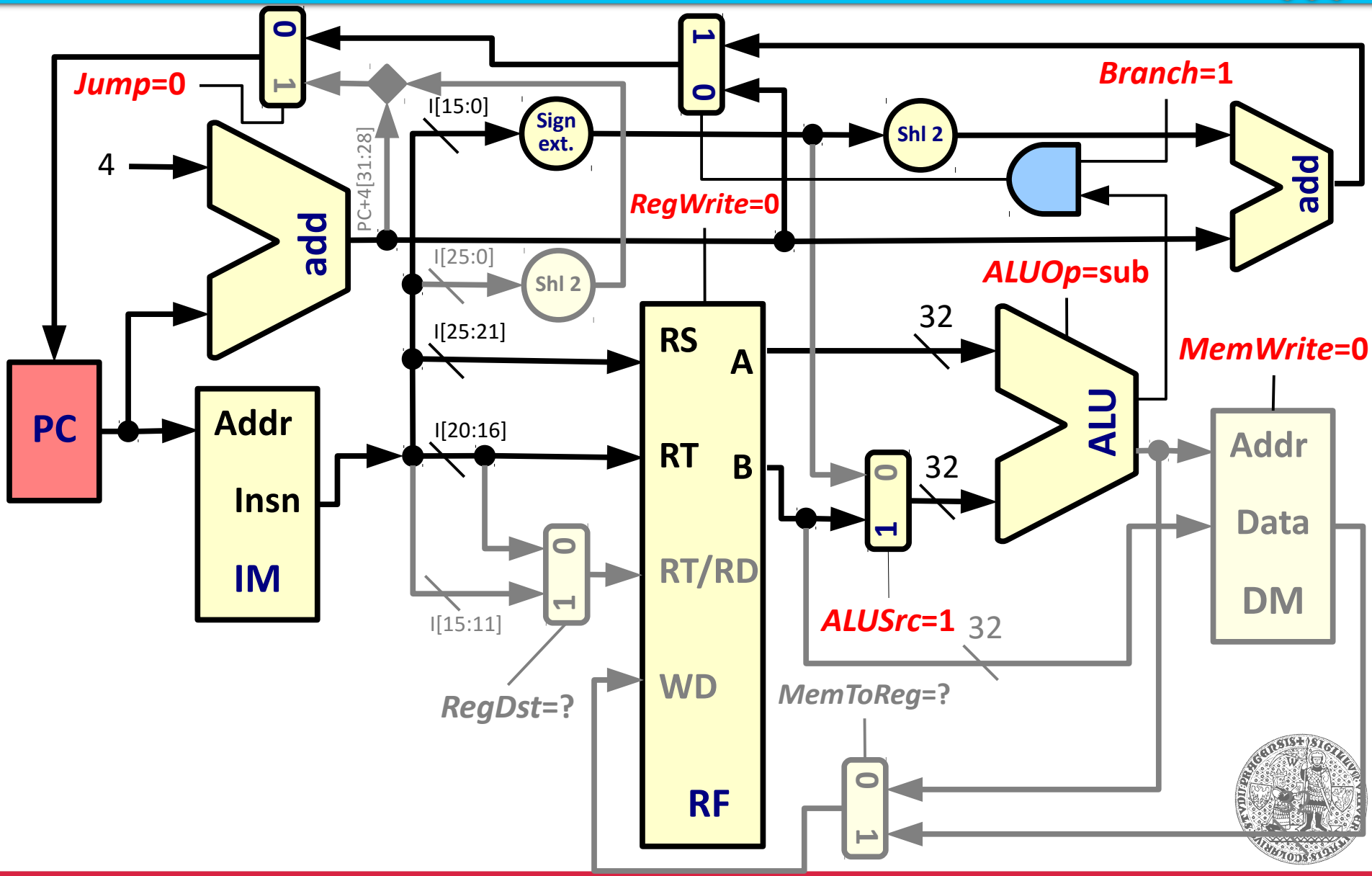
Příklad: Řízení datové cesty pro *add*



Příklad: Řízení datové cesty pro *sw*



Příklad: Řízení datové cesty pro *beq*



Řadič datové cesty

- ***Data path controller***
  - Logický obvod generující řídící signály
  - Hodnoty signálů závisí na operačním kódu instrukce
    - Některé mohou být přímo součástí instrukčního kódu
      - **MIPS:** Část signálů *ALUOp* odpovídá bitům v poli `funct` instrukcí formátu R-type
      - Zjednodušuje implementaci řadiče

Řadič pomocí ROM

- **Řadič s řídící pamětí**
  - Slova v paměti reprezentují hodnoty řídících signálů
  - Hodnota operačního kódu adresuje řádky paměti

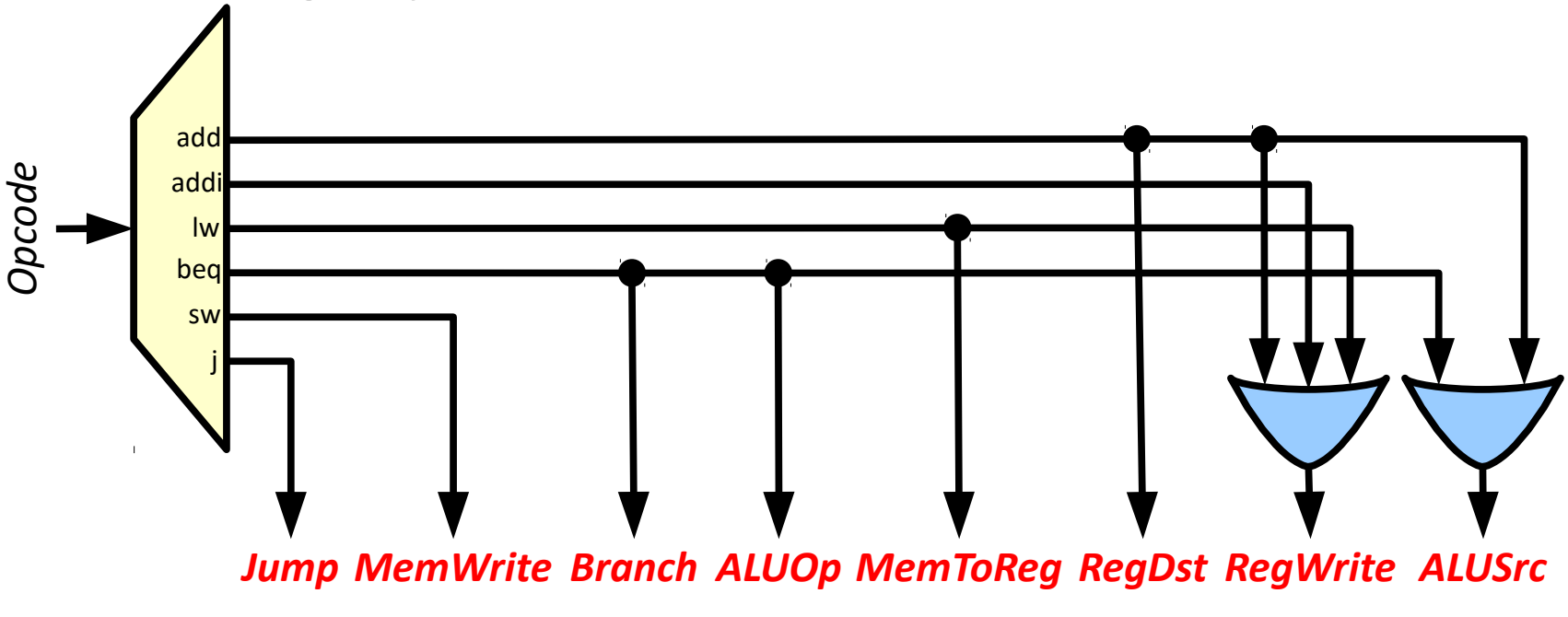
opcode	*Jump*	*Branch*	*RegDst*	*RegWrite*	*MemWrite*	*MemToReg*	*ALUOp*	*ALUSrc*
add	0	0	1	1	0	0	add	1
addi	0	0	0	1	0	0	add	0
lw	0	0	0	1	0	1	add	0
sw	0	0	?	0	1	?	add	0
beq	0	1	?	0	0	?	sub	1
j	1	?	?	0	0	?	?	?

Řadič pomocí ROM (2)

- **Reálný procesor MIPS**
  - Zhruba 100 instrukcí a 300 řídících signálů
    - Kapacita řídící ROM zhruba 30000 bitů (~ 4 KB)
  - Implementační problémy
    - Jak vyrobit ROM rychlejší než je datová cesta

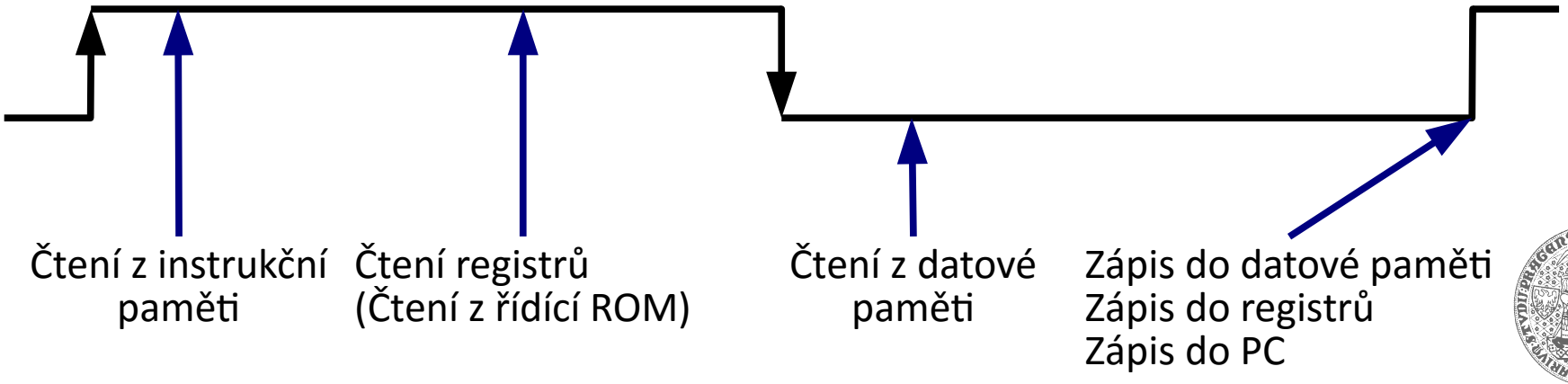
Řadič pomocí kombinačního obvodu

- **Rychlejší alternativa k ROM**
  - Pozorování: Mnoho řídících signálů má málo jedniček nebo nul
  - Obsah ROM lze kompaktně reprezentovat logickými funkcemi



Průběh hodinového cyklu

- **Datová cesta s nepřetržitým čtením**
  - V našem návrhu není na závadu
    - Zápisy (PC, RF, DM) jsou nezávislé
    - V rámci cyklu žádné čtení nenásleduje po zápisu
    - Čtení instrukce (fetch) nepotřebuje řízení
      - Po přečtení instrukce řadič dekóduje operační kód na řídící signály pro zbývající části datové cesty
      - Při změně PC se začne zpracovávat další instrukce



Výkon jednocyklového procesoru

- **Každá instrukce trvá právě 1 takt (CPI=1)**
  - Jednocyklový řadič (řídící ROM nebo kombinační obvod)
  - Obecně nižší taktovací frekvence
  - Délka cyklu odpovídá délce nejdelší instrukce
    - V našem případě *load*
    - Obvykle násobení, dělení nebo floating point operace
  - Datová cesta obsahuje duplicitní prvky
    - Instrukční a datová paměť, dvě sčítačky navíc

Vícecyklová datová cesta

- **Základní myšlenka**
  - Proměnná doba zpracování instrukcí
    - Jednoduché instrukce by neměly trvat stejně dlouho jako složité
    - Perioda hodinového signálu je konstantní → zpracování instrukce rozděleno do více kroků
      - Perioda hodinového signálu odpovídá **délce kroku**
      - Instrukční cyklus vs. strojový cyklus

Výkon vícecyklového procesoru

- **Hrubý odhad**
  - Předpoklady
    - Jednoduché instrukce trvají 10 ns
    - Násobení trvá 40 ns
    - Typický instrukční mix obsahuje v průměru 10 % násobení
  - Jednocyklová datová cesta
    - Perioda hodin 40 ns, CPI=1 → výkon **25 MIPS**
  - Vícecyklová datová cesta
    - Perioda hodin 10 ns, CPI=1,3 → průměrně 13 ns na instrukci, výkon **77 MIPS** (trojnásobné zlepšení)

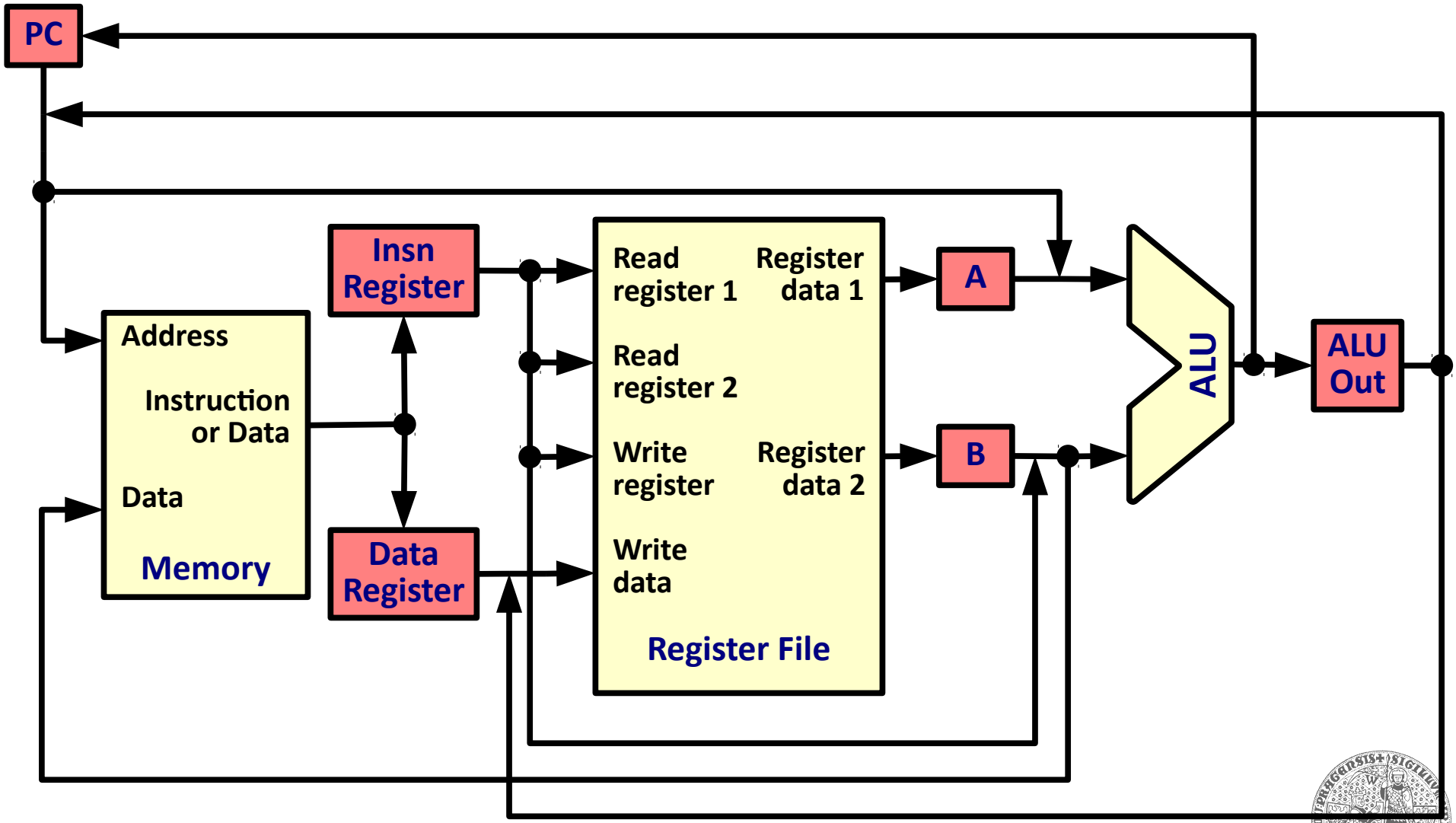
Rozdělení instrukce do kroků

- **Obvyklý instrukční cyklus**
  1. Čtení instrukce
  2. Dekódování instrukce, čtení registrů
  3. Vykonání operace / výpočet adresy / dokončení větvení
  4. Zápis výsledku / přístup do paměti
  5. Dokončení čtení z paměti

Vícecyklová datová cesta (2)

- **Princip implementace**
  - Rozdělení zpracování instrukce do kroků
    - Izolace kroků pomocí registrů pro mezivýsledky
  - Funkce řadiče
    - Realizace návaznosti jednotlivých kroků v datové cestě
    - Některé instrukce mohou některé kroky přeskočit a skončit dříve

Vícecyklová datová cesta (3)



1. krok: Čtení instrukce

- **Současně probíhá**
  - IR ← Memory[PC]
    - Přečtení instrukce do instrukčního registru
  - PC ← PC + 4
    - Posun PC na adresu další instrukce v sekvenci
    - Změna hodnoty PC čtení instrukce neovlivní, protože přečtená instrukce je již v instrukčním registru

2. krok: Dekódování instrukce, čtení reg.

- **Současně probíhá**
  - A ← Reg[IR.rs]
    - Přečtení obsahu zdrojového registru A
  - B ← Reg[IR.rt]
    - Přečtení obsahu zdrojového registru B
  - ALUOut ← PC + (SignExtend(IR.addr) << 2)
    - Výpočet adresy podmíněného skoku
    - Pokud instrukce není skok, výsledek se nepoužije
- **Další kroky se liší podle typu instrukce**

3. krok: Vykonání operace / výpočet adr.

- **Instrukce podmíněného skoku *(konec)***
  - (A == B) ⇒ PC ← ALUOut
    - Cíl skoku vypočítaný v předchozím kroku
- **Instrukce nepodmíněného skoku *(konec)***
  - PC ← PC[31:28] + (IR[25:0] << 2)
- **Aritmeticko-logická operace**
  - ALUOut ← A *funct* B
  - ALUOut ← A *funct* SignExtend(IR[15:0])
- **Přístup do paměti**
  - ALUOut ← A + SignExtend(IR[15:0])
    - Výpočet adresy pro přístup do paměti

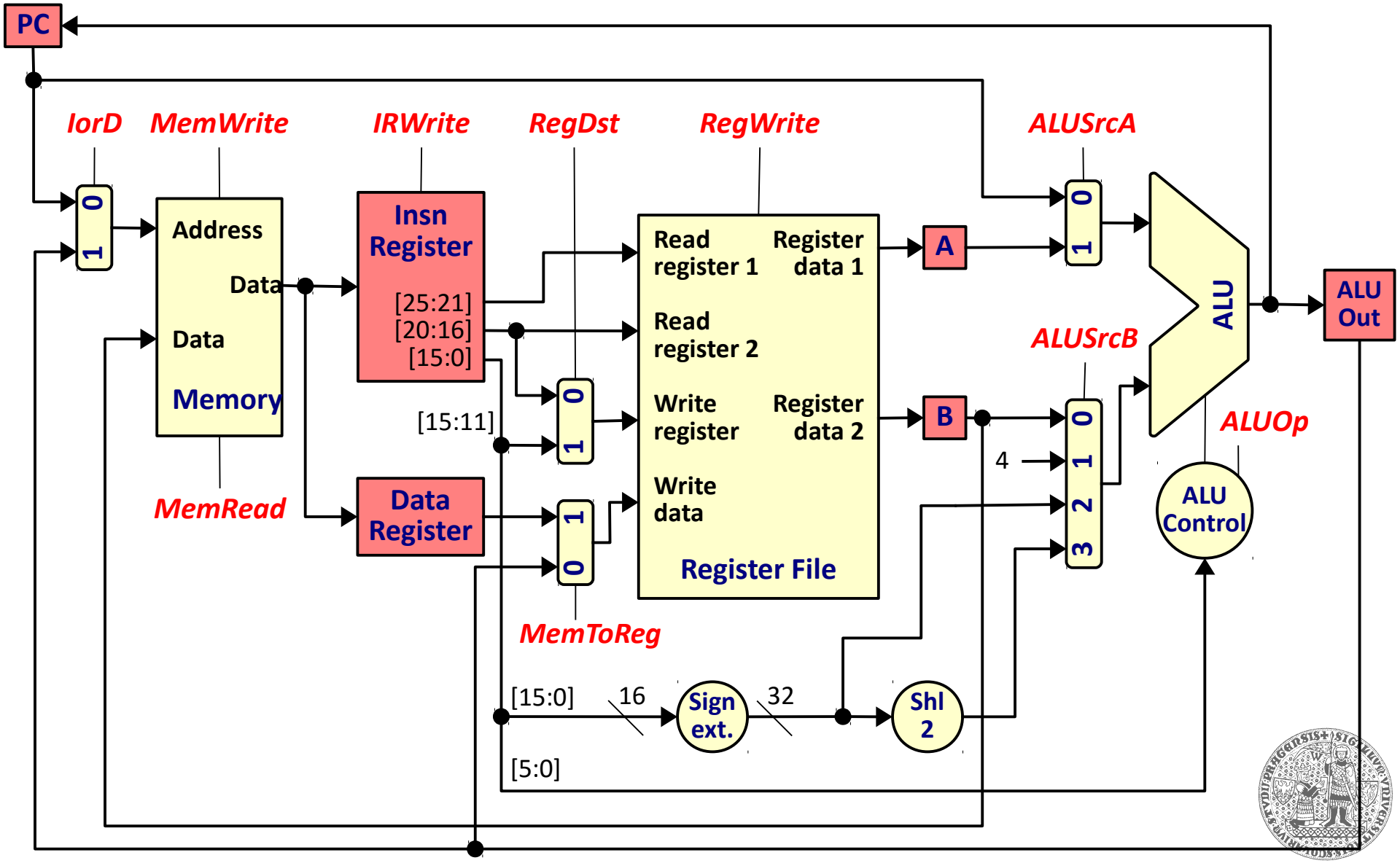
4. krok: Zápis výsledku / přístup do pam.

- **Aritmeticko-logická operace *(konec)***
  - Reg[IR.rd] ← ALUOut
- **Zápis do paměti *(konec)***
  - Memory[ALUOut] ← B
- **Čtení z paměti**
  - DR ← Memory[ALUOut]

5. krok: Dokončení čtení z paměti

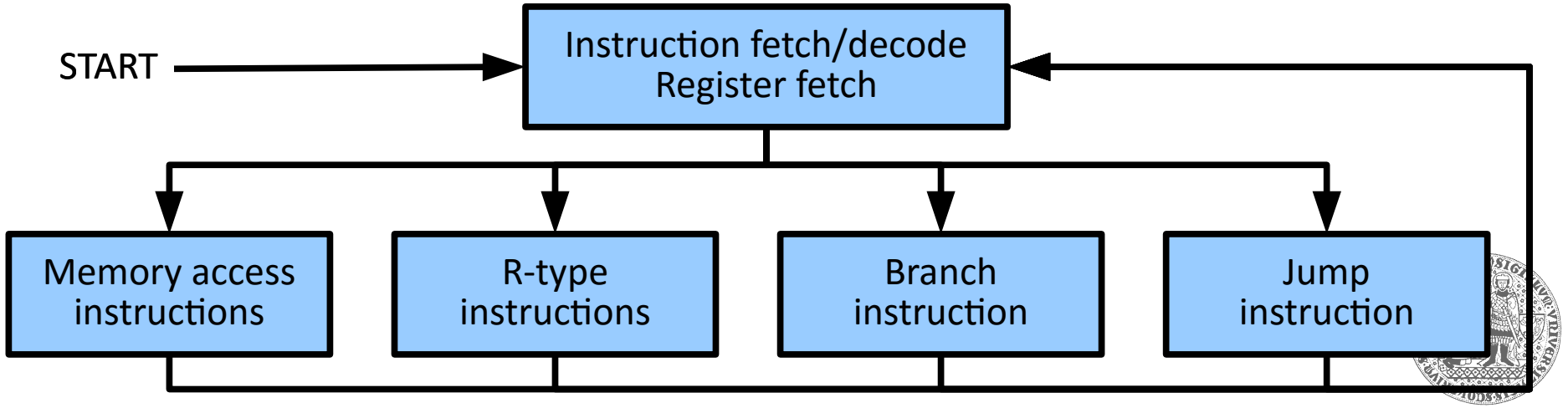
- **Čtení z paměti *(konec)***
  - Reg[IR.rt] ← DR

Implementace vícecyklové datové cesty

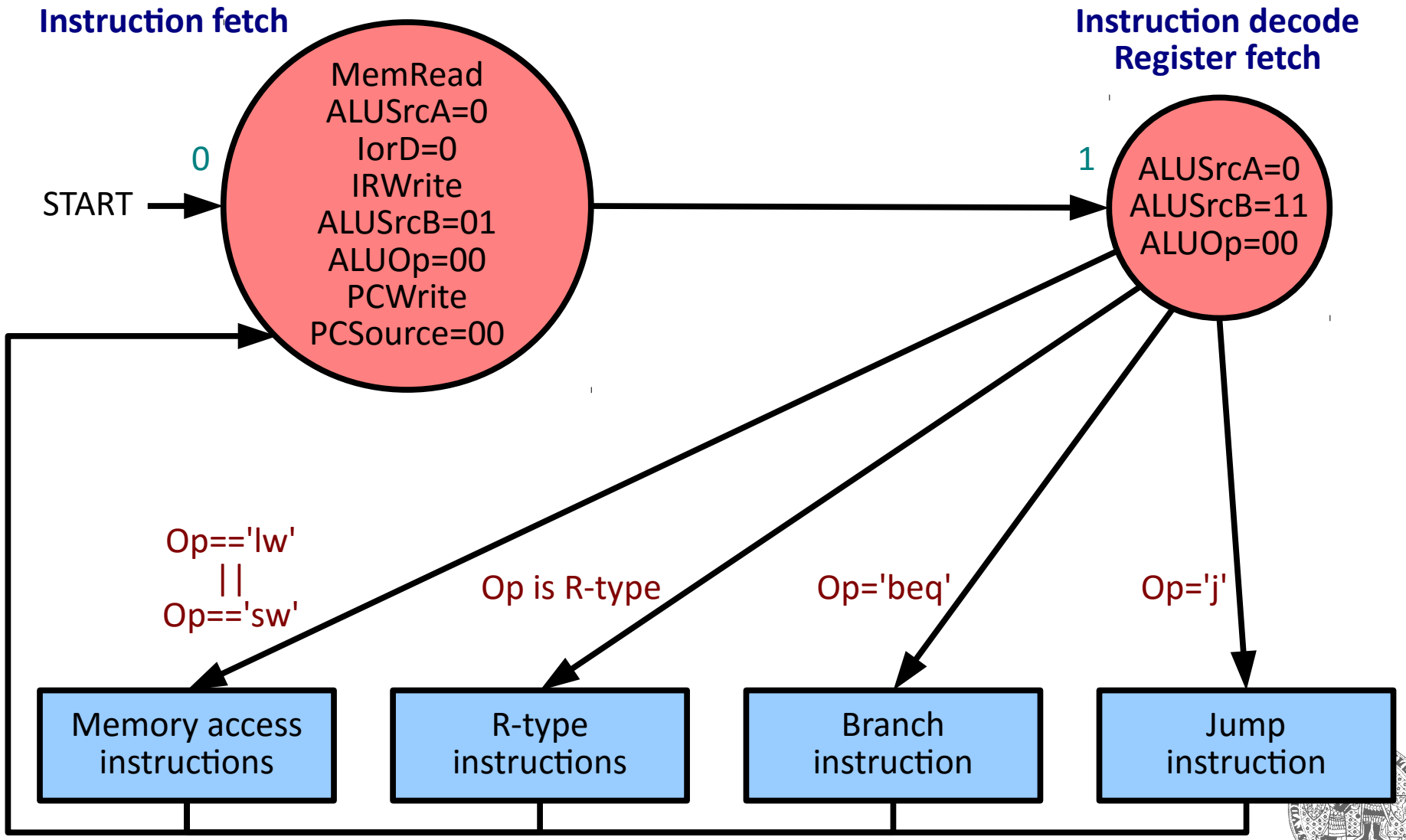


Řízení vícecyklové datové cesty

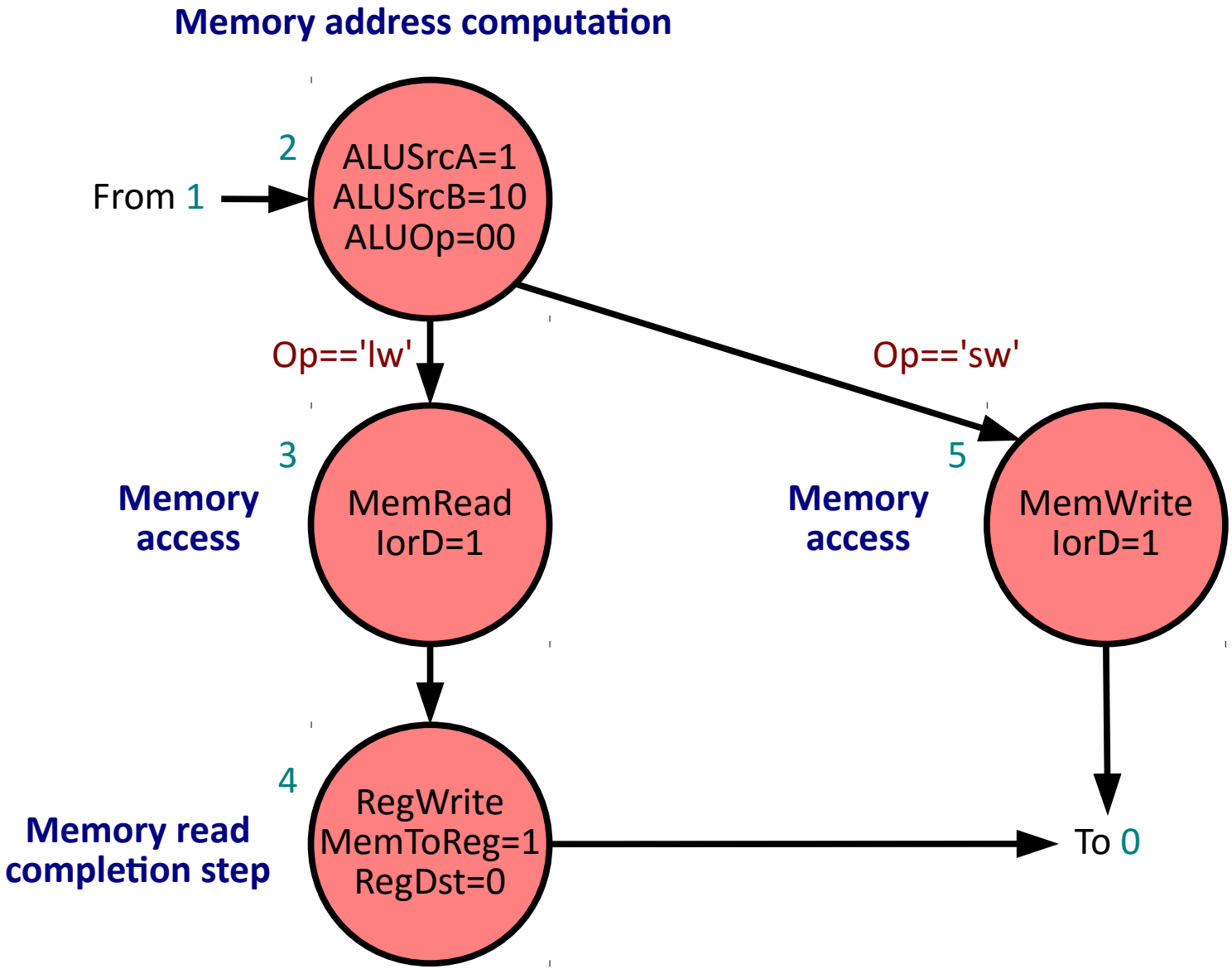
- **Sekvenční proces**
  - Zpracování instrukcí ve více hodinových cyklech
    - Řadič je sekvenční obvod (konečný automat)
    - Aktuální stav řadiče uchováván v paměťovém prvku (stavovém registru)
      - Kombinační logika určuje následující stav, který se do stavového registru zapíše s náběžnou hranou hodinového signálu



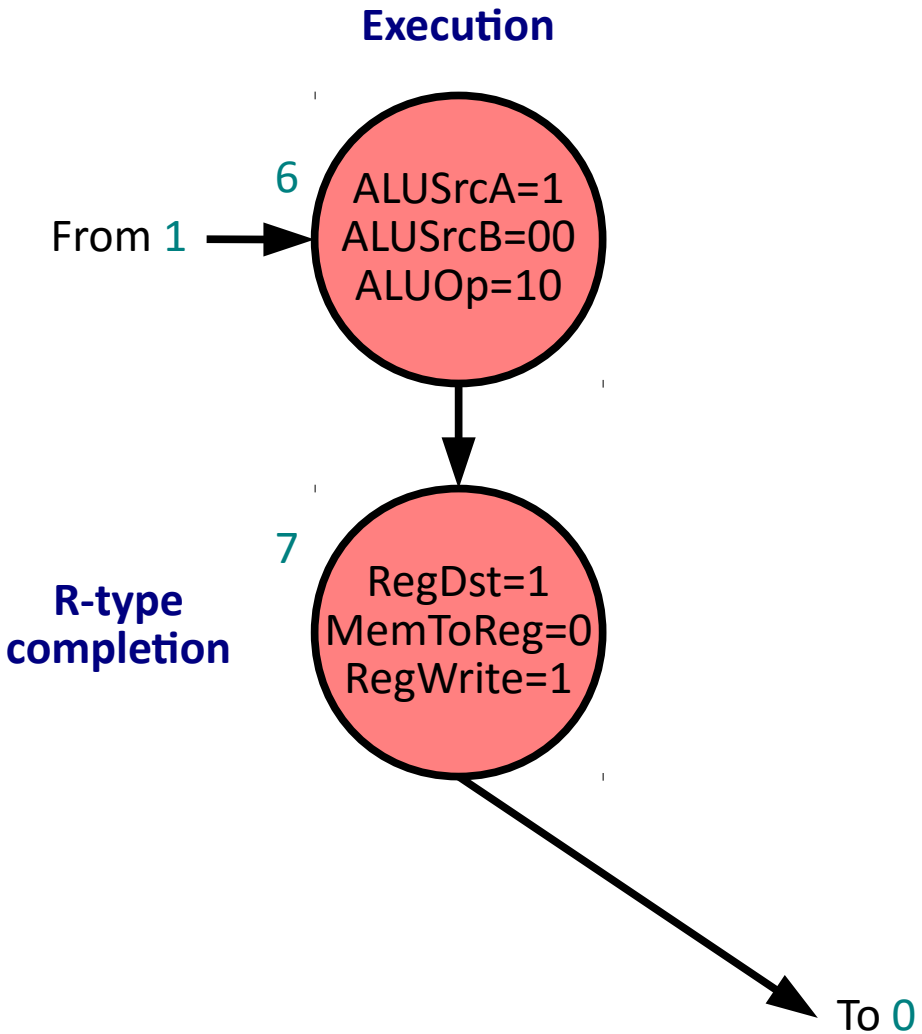
Instruction fetch/decode, Register fetch



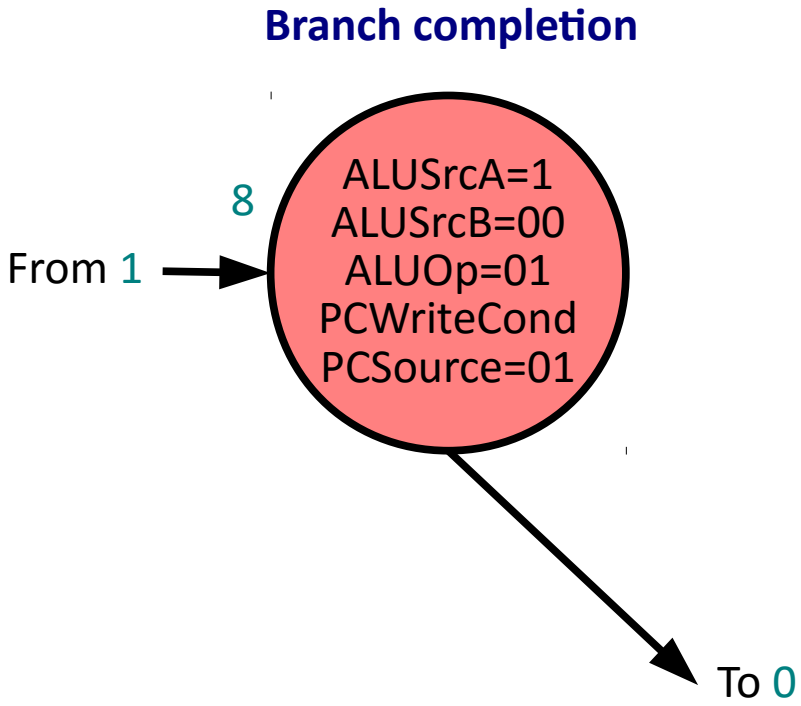
Memory access instructions



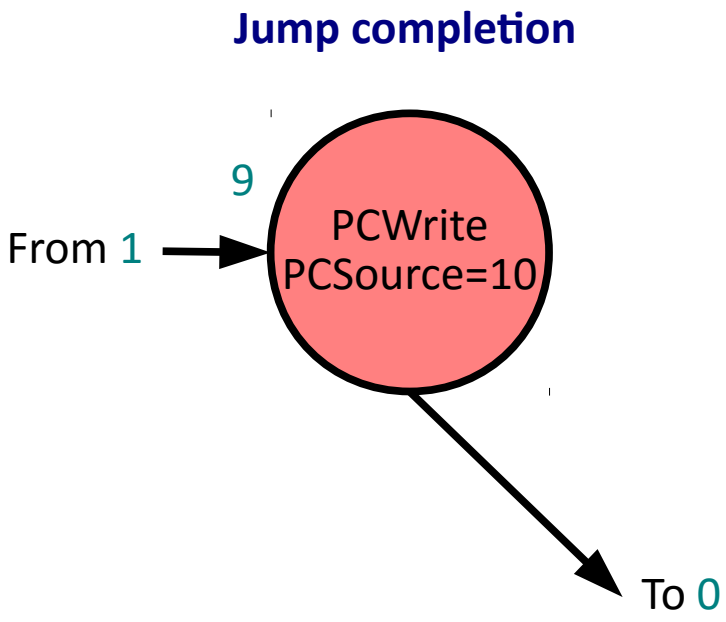
R-type instructions



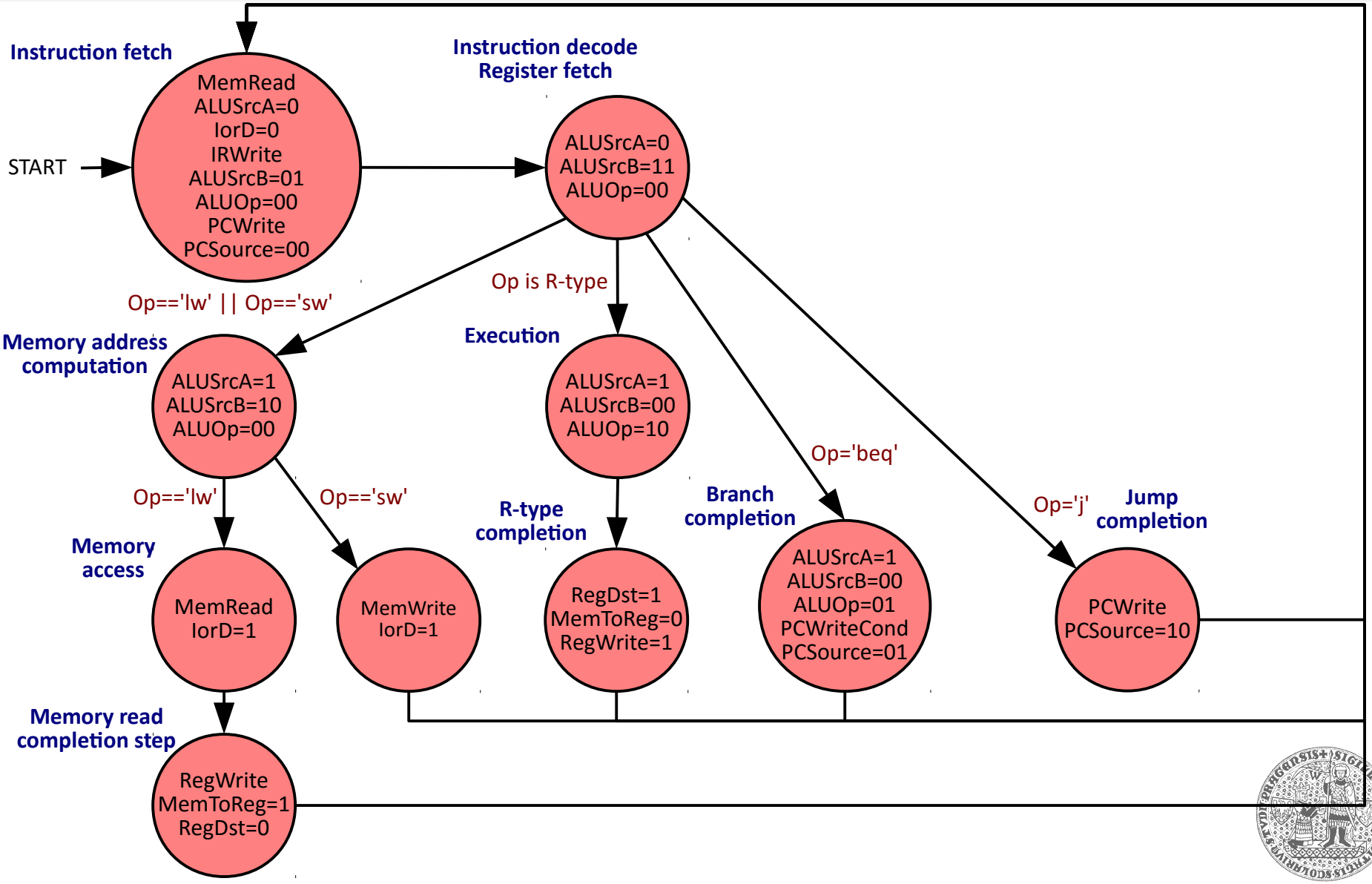
Branch instruction



Jump instruction



Řízení vícecyklové datové cesty (2)



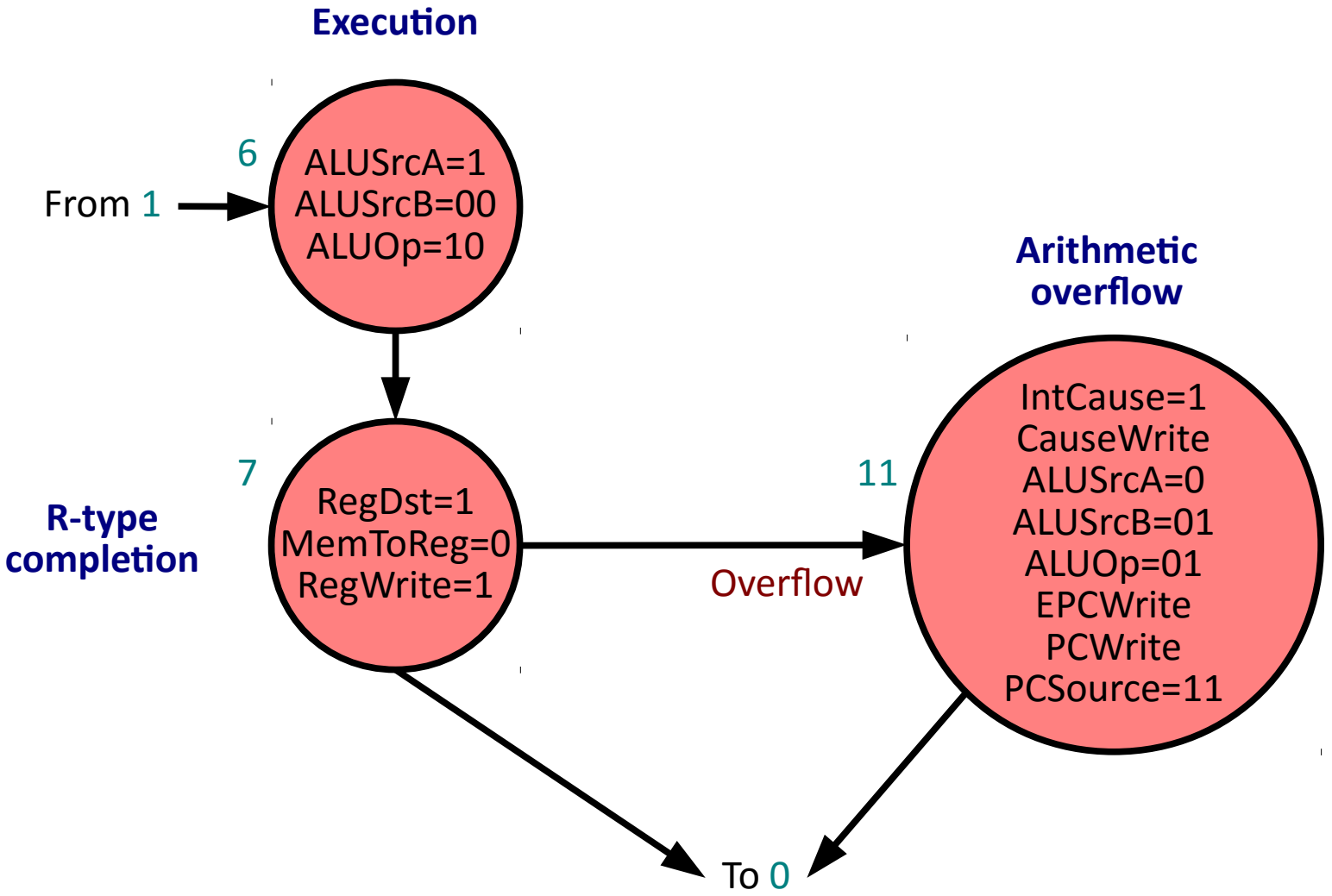
Tok provádění instrukcí

- **Sekvenční a očekávané řízení toku**
  - Běžný kód, podmíněný nebo nepodmíněný skok
- **Neočekávaná změna toku**
  - Vnitřní příčina (*Exception/Trap*)
    - Aritmetické přetečení
    - Nedefinovaná instrukce
    - Nepovolený přístup do paměti
    - Selhání hardwaru
    - Vyvolání služby operačního systému
  - Vnější příčina (*Interrupt*)
    - Požadavek I/O zařízení
    - Selhání hardwaru

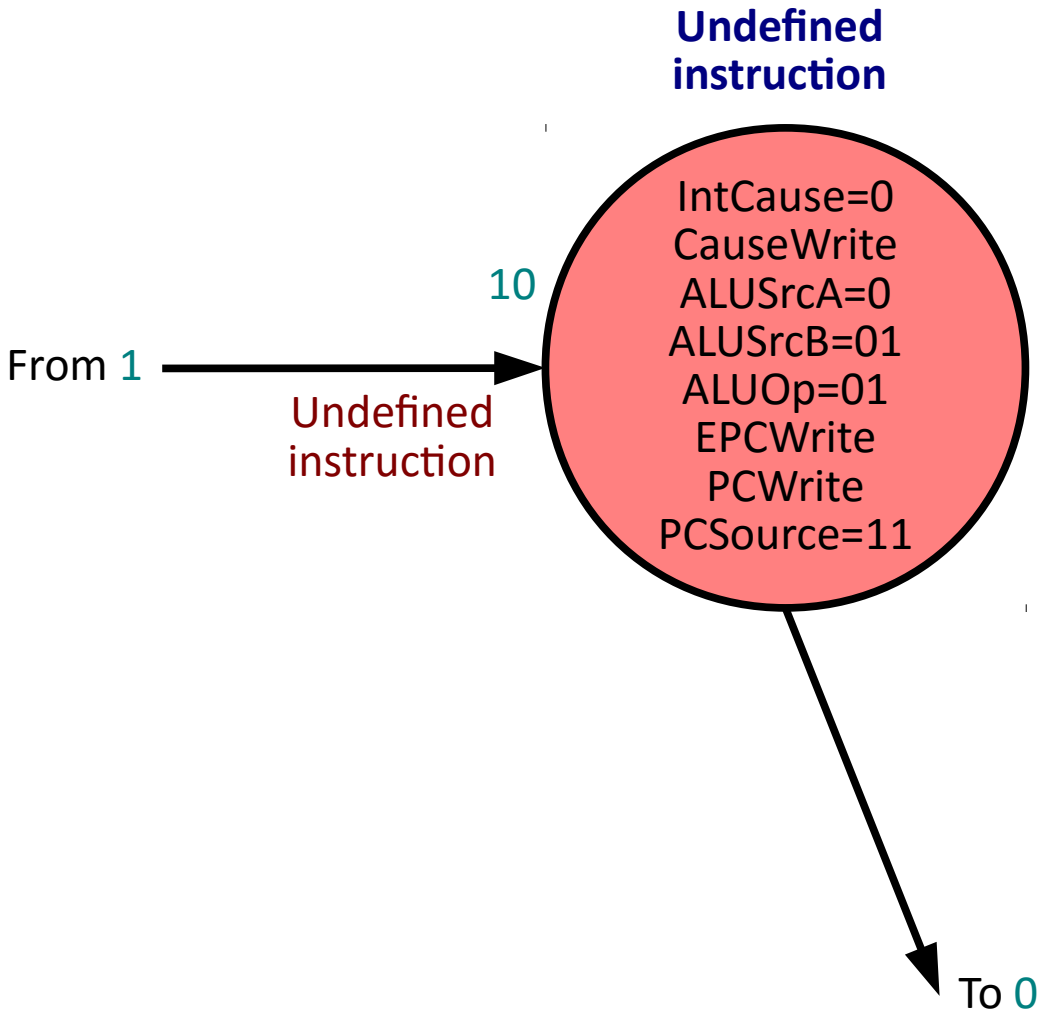
Podpora výjimek a přerušení

- **Hardwarová podpora**
  - Zastavení vykonávání instrukce
    - Zachování korektního stavu procesoru
  - Možnost identifikace příčiny
    - Příznakové bity ve speciálním registru
    - Číslo výjimky
  - Uschování adresy instrukce, při které výjimka nastala
    - Možnost restartovat běh nebo pokračovat v běhu
  - Skok na adresu obslužné rutiny
    - Jedna adresa vs. různé adresy pro různé druhy výjimek

Podpora výjimky přetečení



Podpora výjimky neplatné instrukce



Podpora výjimek a přerušení (2)

- **Softwarová obsluha**
  - Uschování stavu původního výpočtu
  - Zjištění příčiny
  - Obsluha příslušného typu výjimky/přerušení
    - Obsluha I/O zařízení
    - Změna stavu výpočtu
    - Ukončení výpočtu
  - Obnovení stavu původního výpočtu
  - Návrat do původního výpočtu
    - Provedení následující instrukce
    - Restart instrukce, která výjimku vyvolala

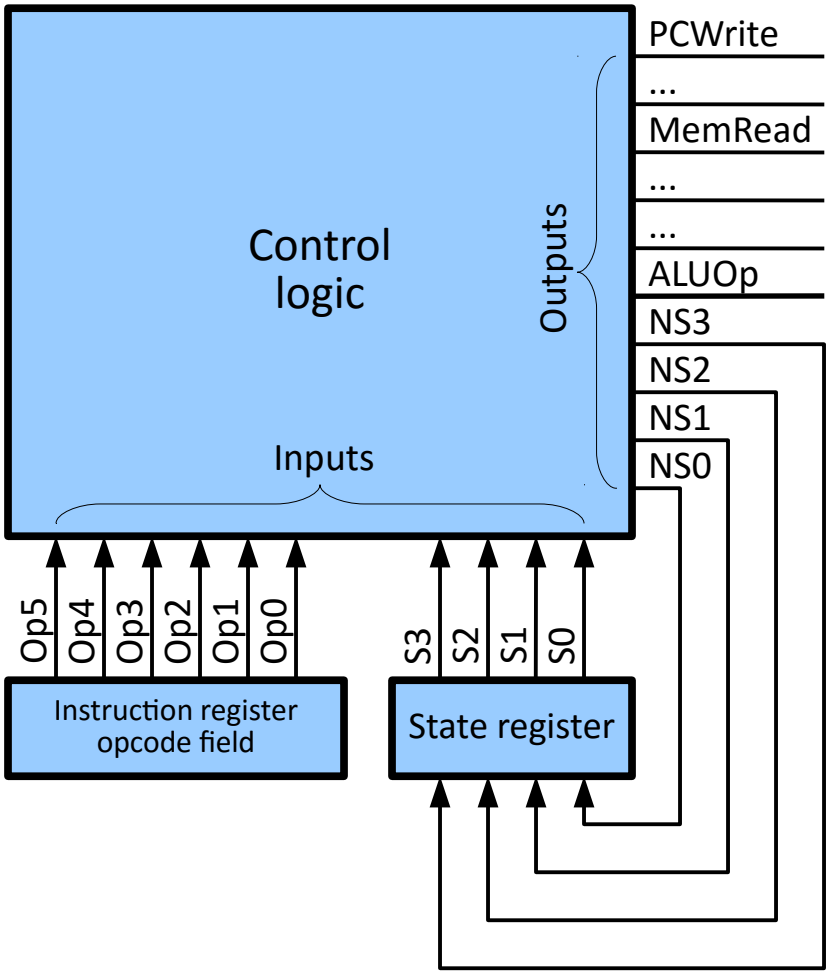
Výkon vícecyklové datové cesty

- **Instrukční mix**
  - 30% load (5ns), 10% store (5ns)
  - 50% add (4ns), 10% mul (20ns)
- **Jednocyklová datová cesta (takt 20ns, CPI = 1)**
  - ***20ns*** na instrukci ⇒ výkon ***50 MIPS***
- **Jednoduchá vícecyklová datová cesta (takt 5ns)**
  - ***CPI≈ (90% × 1) + (10% × 4) = 1.3***
  - ***6.5ns*** na instrukci ⇒ výkon ***153 MIPS***
- **Jemně členěná vícecyklová datová cesta (takt 1ns)**
  - ***CPI≈ (30% × 5) + (10% × 5) + (50% × 4) + (10% × 20) = 6***
  - ***6ns*** na instrukci ⇒ výkon ***166 MIPS***

Realizace vícecyklového řadiče

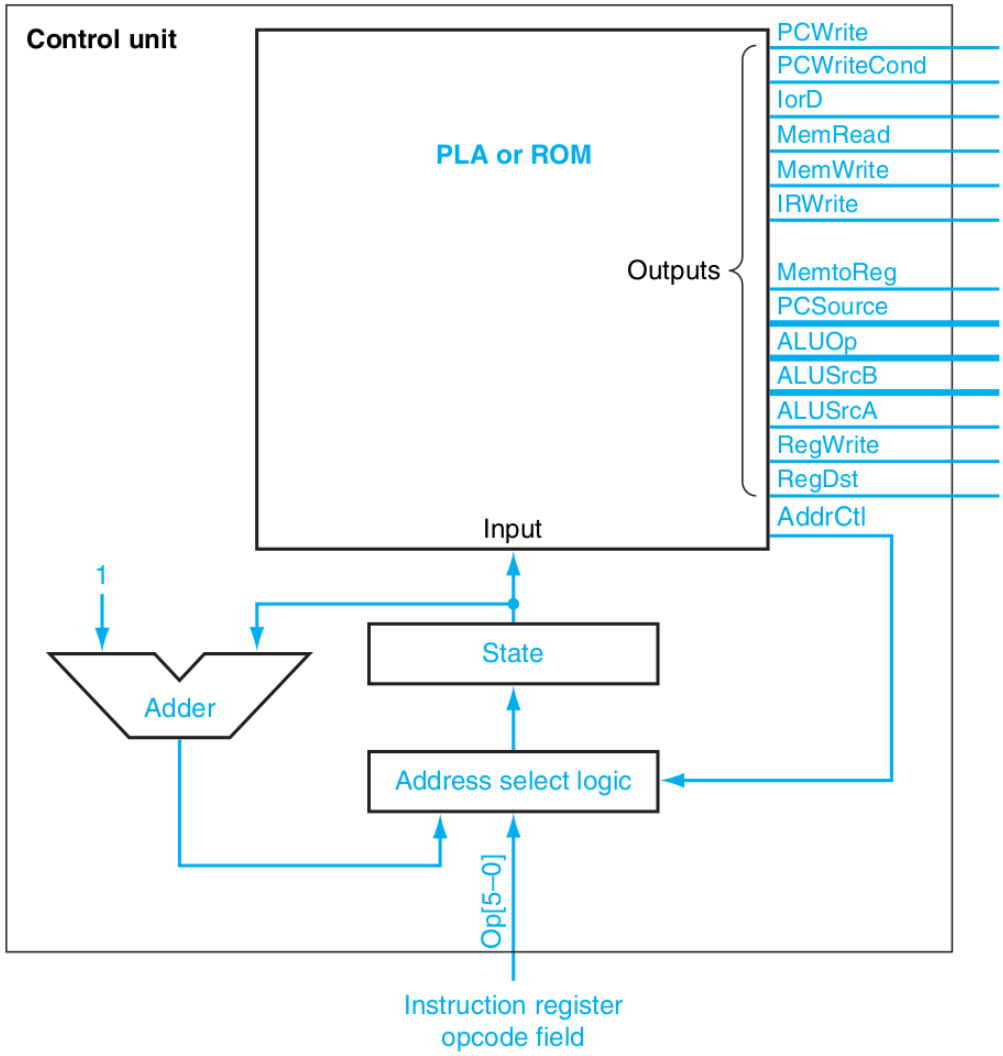
- **Realizace konečného automatu**
  - Stav + podmínky = paměť + logika = sekvenční obvod
    - Konkrétní realizace závisí na reprezentaci vnitřního stavu
    - Obvodové řešení
      - Posuvný řetězec klopných obvodů
      - Stavový registr, kombinační logika
    - Paměť + jednoduchý sekvenční obvod
      - Mikroprogramování, nanoprogramování

Stavový registr + kombinační logika

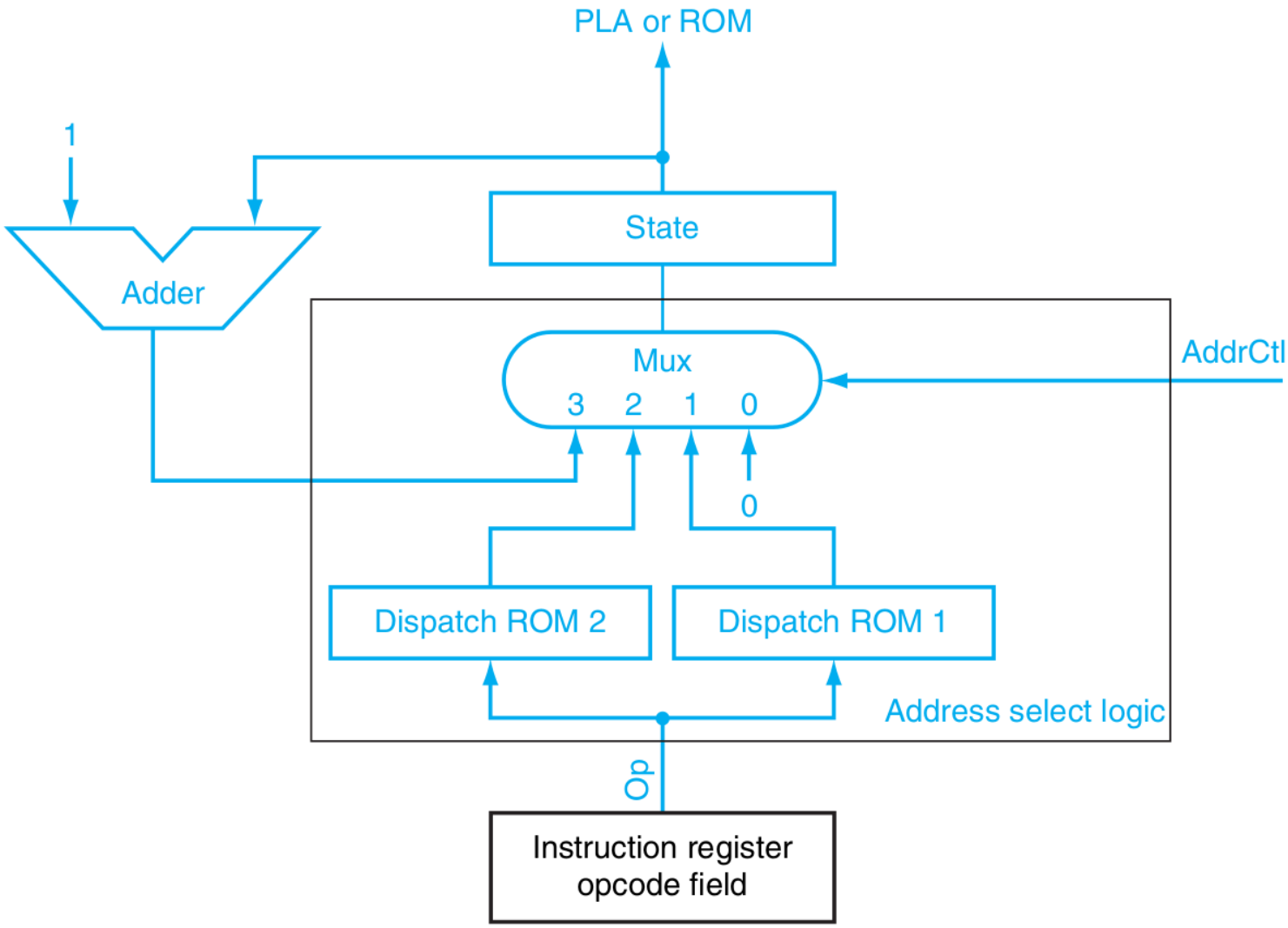


- **Řídící logika**
  - Kombinační obvod
  - ROM, FPGA

Použití čítače pro následující stav

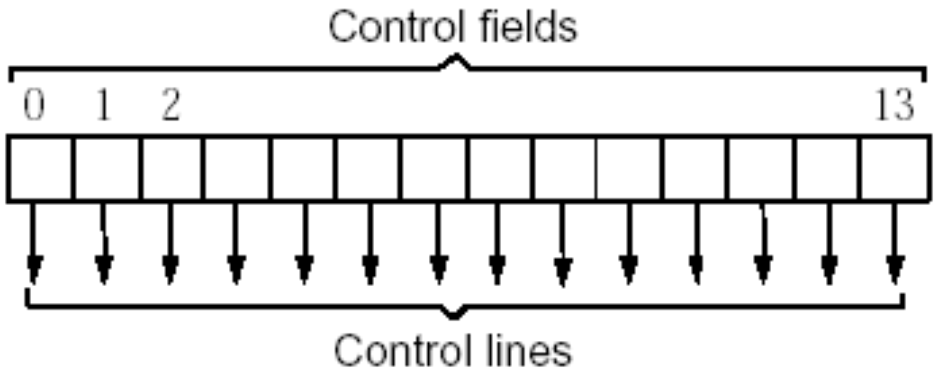


Logika pro výběr adresy



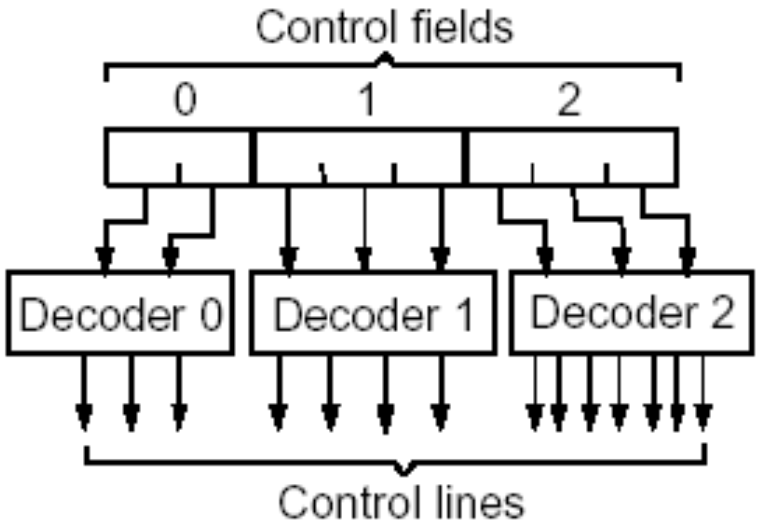
Horizontální formát mikroinstrukcí

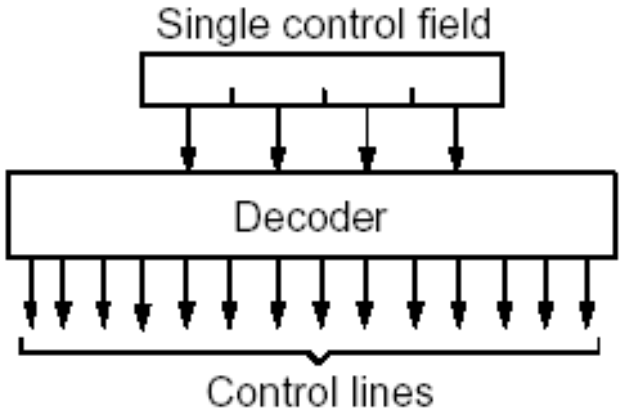
- **Přímá reprezentace řídících signálů**
  - Paměť mikroinstrukcí obsahuje přímo hodnoty řídících signálů
    - Není potřeba dekódovat (rychlost)
    - Libovolná kombinace (pružnost)
    - Velké prostorové nároky



Vertikální formát mikroinstrukcí

- **Kódovaná reprezentace řídících signálů**
  - Mikroinstrukce obsahují jen čísla platných kombinací řídících signálů
    - Tato čísla se dekódují samostatným dekodérem nebo dekodéry (zpomalení, omezení pružnosti)

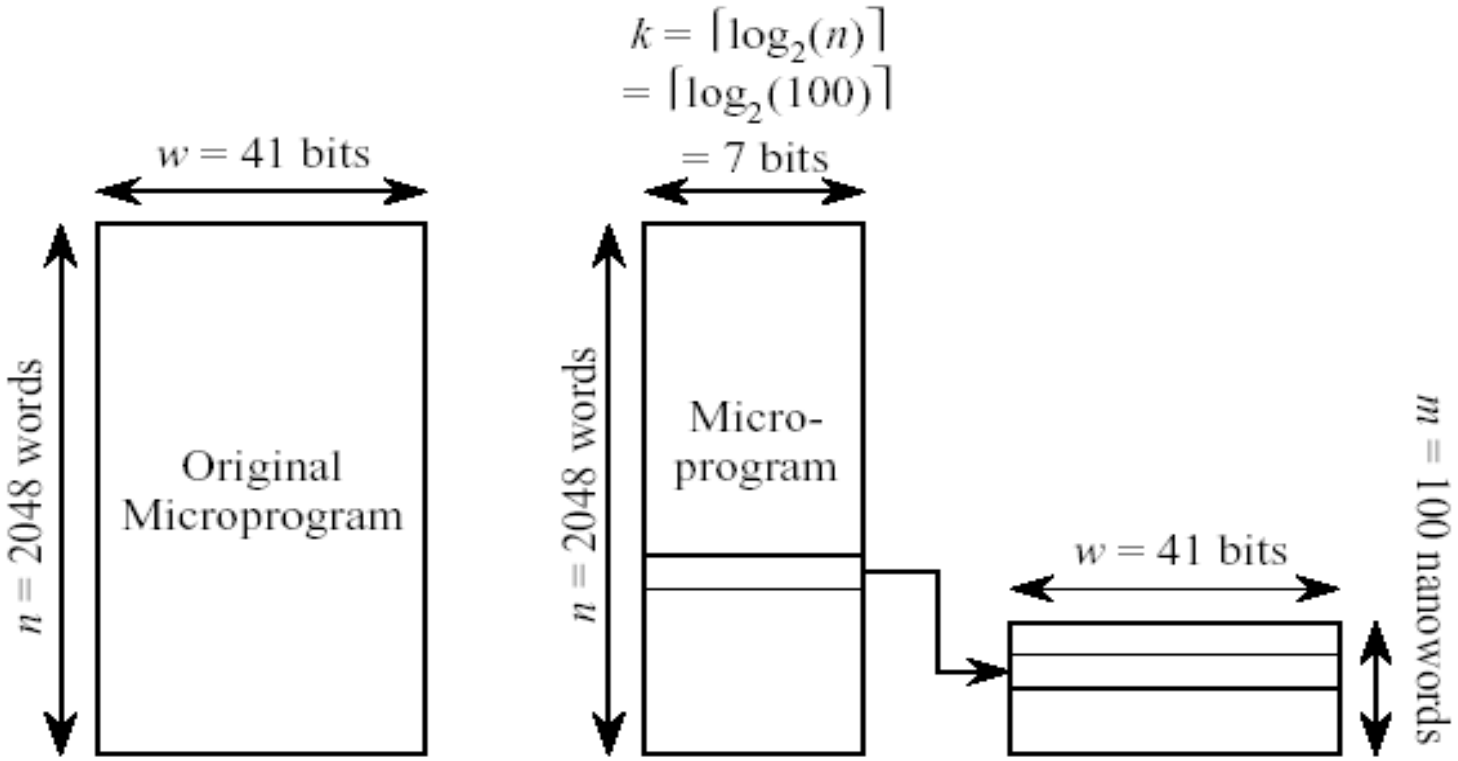




Nanoprogramování

- **Kombinace horizontálního a vertikálního kódování**
  - Mikroprogramová paměť obsahuje jen čísla platných kombinací řídících signálů (vertikální formát)
  - Převod na horizontální formát se nerealizuje pomocí fixního dekodéru (kombinačního obvodu), ale pomocí další paměti
  - Výrazně redukuje prostor potřebný k uložení mikroprogramu, ovšem za cenu nižší rychlosti

Srovnání mikro- a nanoprogramování



Total Area = $n \times w$ = $2048 \times 41 = 83{,}968$ bits

Microprogram Area = $n \times k = 2048 \times 7$
$= 14{,}336$ bits
Nanoprogram Area = $m \times w = 100 \times 41$
$= 4100$ bits
Total Area = $14{,}336 + 4100 = 18{,}436$ bits