

Návrh počítačových systémů

Princip činnosti procesoru

INP 2025

FIT VUT v Brně



Čím se budeme zabývat v INP

- Budou nás zejména zajímat **číslicové počítače s jedním (jednojádrovým) procesorem**:
 - funkce počítače
 - struktura – propojení funkčních bloků,
 - organizace – součinnost, chápána dynamicky, časové řízení bloků,
 - realizace – návrh a struktura bloků (např. ALU, řadič, cache, apod.)
 - měření výkonnosti.
- Reprezentace dat, kódy
- Principy a algoritmy základních i složitějších aritmetických operací
- Paměti – typy, paměťová hierarchie, řízení, ...
- Sběrnice a periferní zařízení
- Spolehlivost
- Úvod do paralelních architektur
- Cvičení:
 - obvodová realizace procesoru a jeho bloků
 - Jazyk VHDL
 - příklady k tématům z přednášek

Počítač (podle Wikipedie)

- **Počítač** je v informatice elektronické zařízení, které zpracovává data pomocí předem vytvořeného programu. Současný počítač se skládá z hardware, které představuje fyzické části počítače (procesor, klávesnice, monitor atd.) a ze software (operační systém a programy). Počítač je ovládán uživatelem, který poskytuje počítači data ke zpracování prostřednictvím jeho vstupních zařízení a počítač výsledky prezentuje pomocí výstupních zařízení. V současnosti jsou počítače využívány téměř ve všech oborech lidské činnosti.
- Analogový počítač vs číslicový počítač
- Historie – viz např. Wikipedie

První elektronické počítače

- **ENIAC** (Electronic Numerical Integrator and Calculator)
 - první elektronický (elektronkový) počítač na světě
 - postaven na University of Pennsylvania
 - postaven během druhé světové války, ale informace o něm byla zveřejněna až v roce 1946
 - programoval se drátovými propojkami a přepínači, data se zadávala pomocí děrných štítků
 - Příkon: 150 kW; 5000 sčítání/s; plocha 167 m²; hmotnost 27 t
- **EDVAC** (Electronic Discrete Variable Automatic Computer)
 - dokončený na téže univerzitě v roce 1951
 - řízen programem uloženým v paměti (5,5 kB)
 - Jeho koncepce se stala nejrozšířenější a nejznámější počítačovou architekturou. Poněkud neprávem se připisuje americkému matematikovi maďarského původu John von Neumannovi, přestože hlavními osobami projektu byli J. Presper Eckert a John Mauchly.
 - Příkon: 56 kW; 1160 sčítání/s; plocha 45 m²; hmotnost 7,8 t
 - Pracoval 20 hodin denně a průměrně 8 hodin bez poruchy.

Současné počítače - příklady

- **Notebook**

- Procesor: Intel Core i7, 10870H Comet Lake, 8 jader, 2,2 GHz, 14 nm, TDP 45 W
- Displej 15.6", 1920 × 1080 144Hz
- RAM 16 GB DDR4
- NVIDIA GeForce RTX 3060 6 GB
- SSD 512 GB
- Hmotnost 2,46 kg



- **Paralelní počítač na jednom čipu: Intel Xeon Phi**

- 61 jader (Pentium)
- 244 vláken
- 16 GB (GDDR5)
- 1,2 TFLOP/s max. výkonnost
- příkon 225 W



- **iPhone 13 Pro**

- procesor Apple A15 Bionic (založený na 64-bit ARM), 6-jádrový, Neural Engine, 5 nm TSMC
- 8 GB RAM a 1000 GB storage
- OLED displej 6,7"
- iOS 15
- Baterie 4352 mAh



- **Superpočítač Karolina (IT4I, Ostrava)**

- 15,7 PFLOP/s teoretický výpočetní výkon
- 720 uzlů – 2x AMD 7H12, 64 jader, 2,6 GHz, celkem 92160 jader
- 72 uzlů, každý má 8 GPU akceleratorů
- a další uzly, celkem 831 výčetních uzlů
- příkon 800 kW
- 30 TB úložiště
- 69. stroj v r. 2021 dle top500.org
- <https://www.it4i.cz/infrastruktura/karolina>



Algoritmus

(viz kurz Základy programování)

- **Algoritmus** je přesně definovaná konečná posloupnost příkazů (které jsou vybírány z předem definované konečné množiny elementárních příkazů), jejichž prováděním pro každé přípustné vstupní hodnoty získáme po konečném počtu kroků odpovídající hodnoty výstupní. Intuitivně algoritmem rozumíme postup, který nás dovede k řešení úlohy.

Dvě základní implementace algoritmu

- **program** (SW) pro procesor
 - výhody: univerzalita/flexibilita, cena
 - nevýhody: výkonnost nemusí být dostatečná
- **aplikačně-specifický číslicový obvod** (HW)
 - výhody: umožňuje optimální využití technických prostředků (optimalizace rychlosti, plochy, spotřeby apod.)
 - nevýhody: obvykle vysoká cena, dlouhá doba návrhu

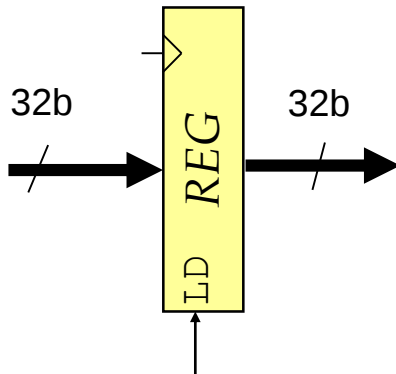
Jak se implementují datové a řídicí struktury?

Datové struktury: SW vs HW

- SW - příklad

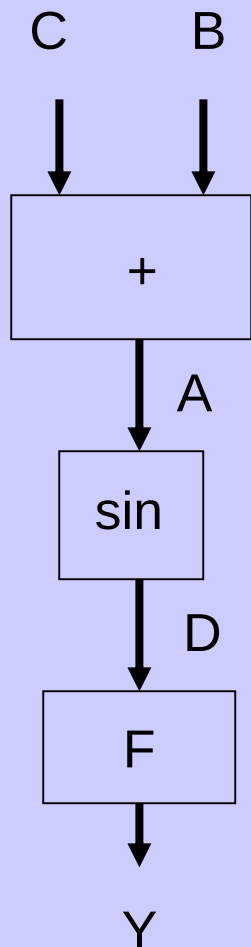
```
int x; // datový typ int je v daném procesoru např. 32b  
x = 34; // přiřazení do proměnné  
y = x;   // čtení proměnné
```

- HW - příklad



Sekvence:

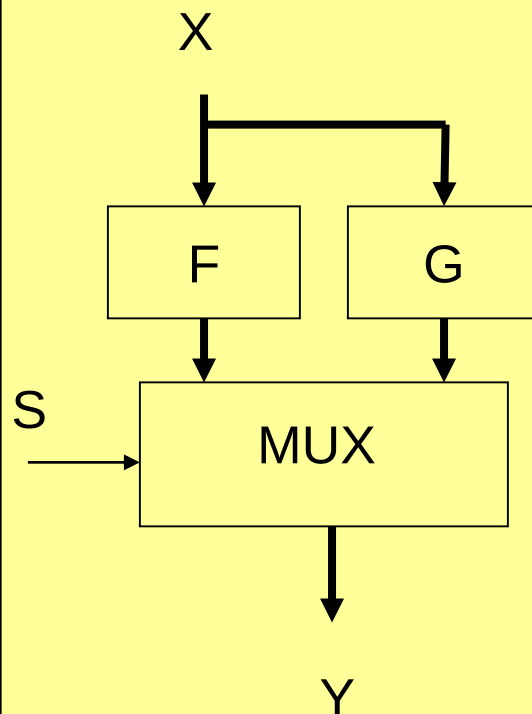
```
A = B + C;  
D = sin(A);  
Y = F(D);
```



Řídicí struktury: SW vs HW

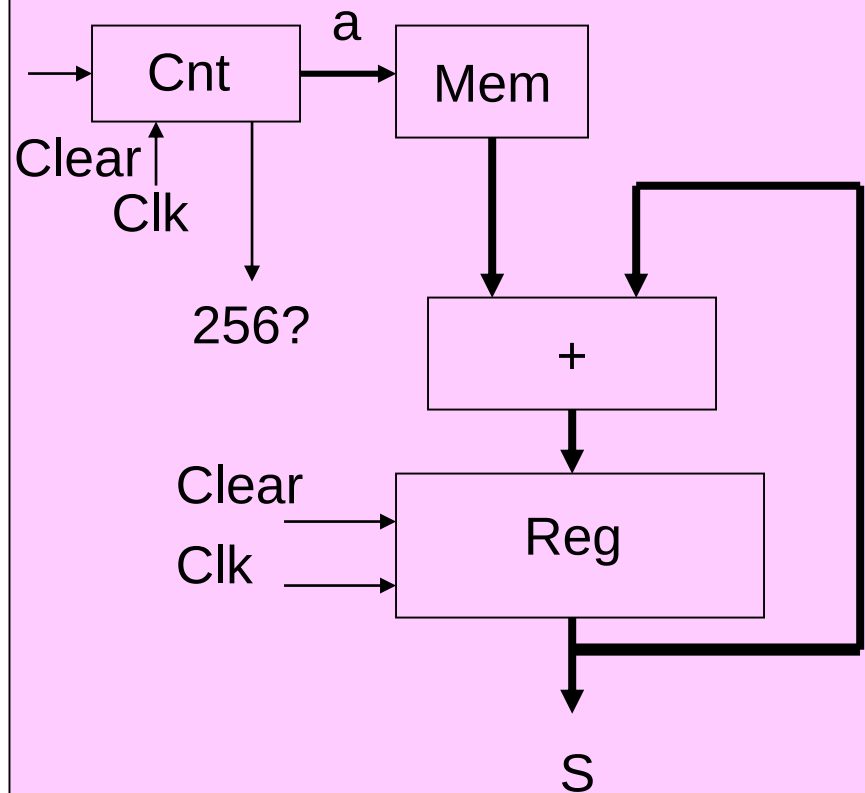
Selekce:

```
If (S == 0)  
then Y = F(X)  
else Y = G(X);
```



Iterace:

```
a = 0;  
S = 0;  
while (a < 256) do  
{  
    S = S + Mem[a];  
    a++;  
}
```



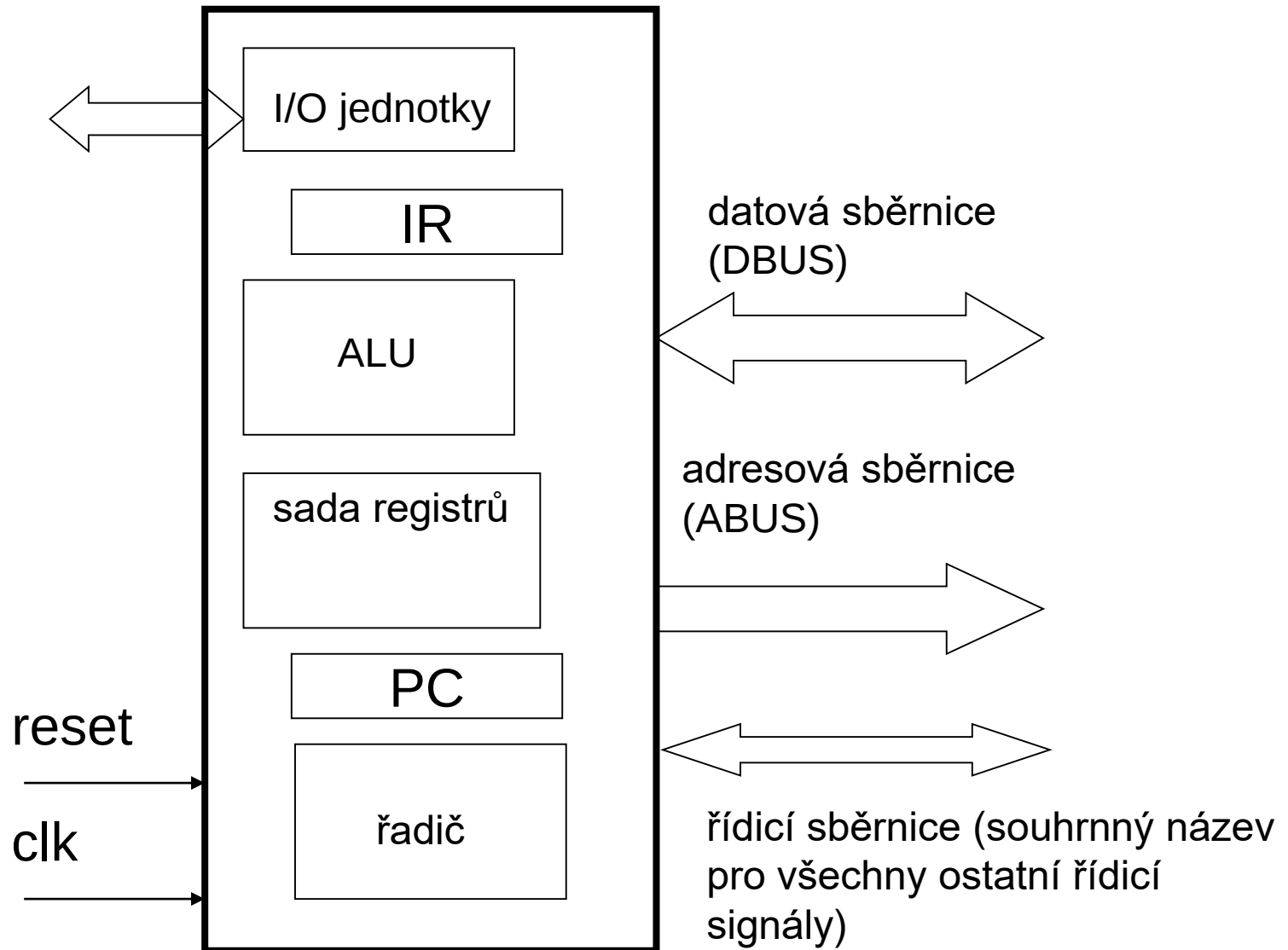
Procesor

- **Procesor:**
 - obvodová realizace algoritmu, který dokáže vykonávat libovolné **programy** (algoritmy), sestavené z předem definovaných elementárních operací (**instrukcí**). Je to tzv. **univerzální výpočetní stroj**.
 - (obvykle) sekvenční a (obvykle) synchronní číslicový obvod
- **Typická činnost:**
 - Procesor vykoná **program**, který je uložen od zadané **adresy** v **paměti**. Program transformuje **data**, uložená na zadané adrese v paměti, na jiná data, jejichž nové umístění musí být rovněž specifikováno.
 - Pokud má procesor **I/O jednotku**, může rovněž při zpracování využívat informace ze svých vstupních portů a produkovat data na výstupní porty.
- **Hlavní komponenty**
 - **Programový čítač** (PC) určuje adresu, kde se nachází následující instrukce. Po resetu dochází k inicializaci procesoru a mj. je definován obsah PC (např. PC = 0003h).
 - IR – **instrukční registr** – uchovává právě zpracovávanou instrukci
 - **Řadič řídí** činnost procesoru (konečný automat - FSM).
 - **ALU** – aritmeticko-logická jednotka slouží k provádění matematických operací.
 - **I/O jednotka** – umožňuje vstup a výstup dat do/z procesoru (obvykle do jeho registrů) i jinak než pouze skrz paměť.
 - Sada registrů

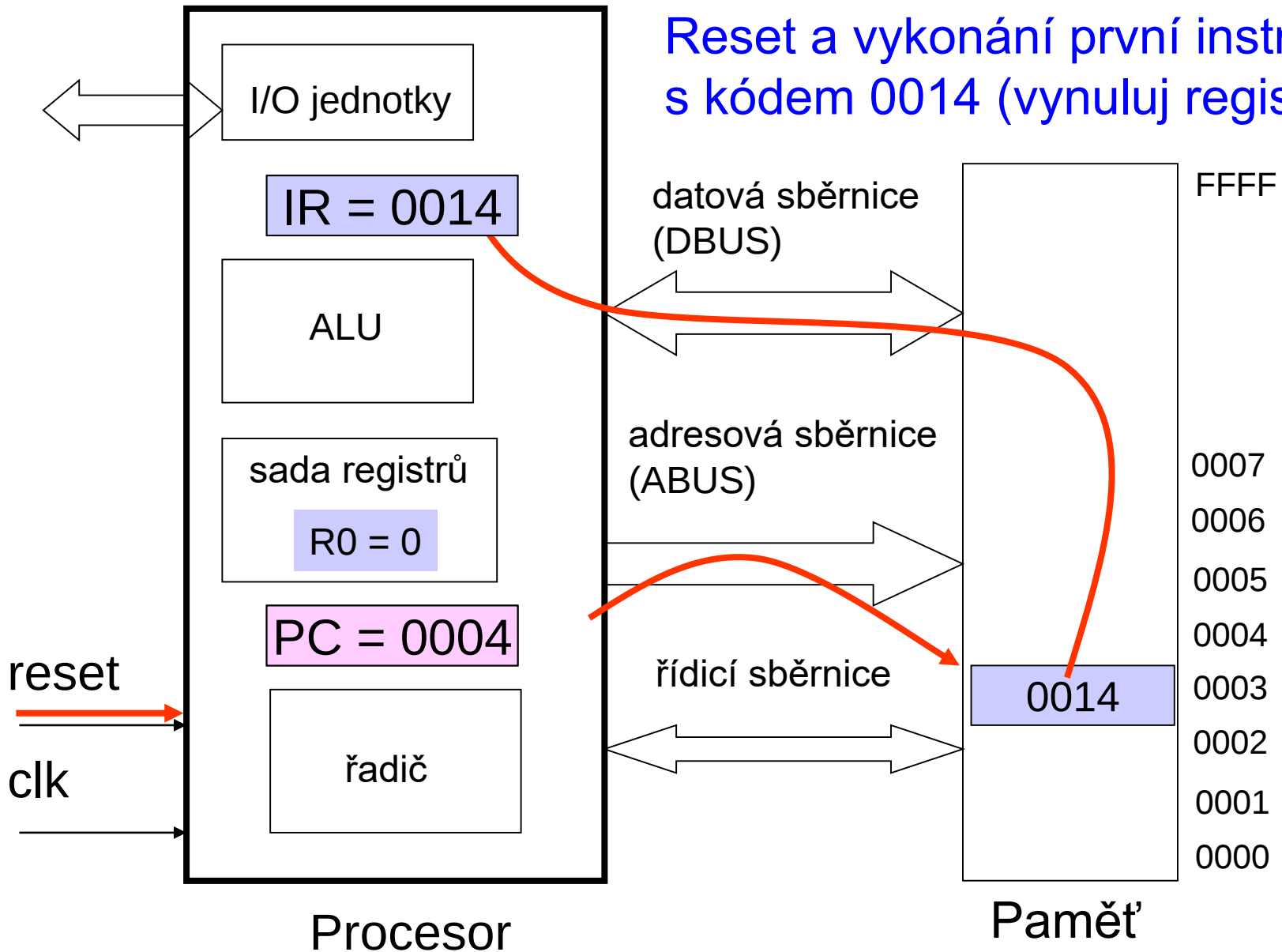
Programy a data

- **Programy** jsou uloženy v paměti jako posloupnosti instrukcí.
- Program se vykonává sekvenčně (až na výjimky, např. skokové instrukce)
- **Data** jsou uložena rovněž v paměti.
- **Harvardská** koncepce procesoru
 - Paměť dat a paměť programu jsou odděleny, tj. je možné současně např. načíst instrukci a zapisovat data.
- **Von Neumannova** (princetonská) koncepce procesoru
 - Společná paměť pro data a programy, nelze např. současně načítat instrukci a načítat data.

Procesor a jeho rozhraní



Reset a vykonání první instrukce s kódem 0014 (vynuluj registr R0)



Po resetu se nastaví PC=0003. Adresa určená obsahem PC se vystaví na ABUS. Z této adresy se přečte kód instrukce (0014) a po DBUS se pošle do instrukčního registru IR. Instrukce se dekoduje a vykoná (vynuluje se registr R0) a zvýší se PC o 1.

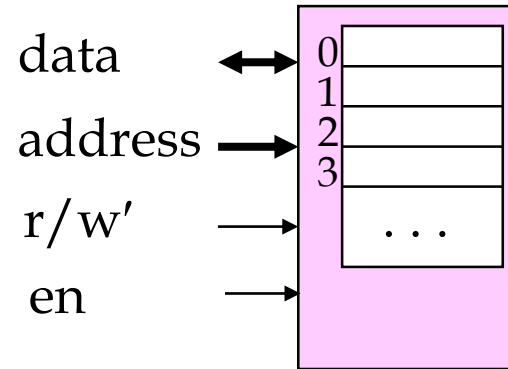
Vykonání instrukce

- Adresa instrukce, která se má vykonat, je uložena v PC.
- Instrukce se vykonává obvykle v několika hodinových taktech.
- Procesor musí zajistit následující:
 - Adresa určená obsahem PC je vystavena na adresovou sběrnici a je iniciováno čtení z paměti.
 - Paměť dodá na datovou sběrnici kód instrukce, která se má vykonat.
 - Instrukce je uložena do IR (instrukčního registru). Tyto tři body se označují jako načtení instrukce – **fetch (F)**.
 - Instrukce je dekodována (tzn., že jsou nastaveny řídicí signály pro komponenty procesoru, které se podílí na vykonání instrukce – je např. vybrána operace sčítání a operandy ALU). Tato fáze se nazývá **instruction decode (ID)**
 - Pokud instrukce vyžaduje pro své vykonání operand z paměti, musí být z příslušné adresy načten tento operand.
 - Instrukce je vykonána (např. provede se sčítání), jsou nastaveny příznaky (např. přetečení). Tato fáze se nazývá **Execute (E)**.
 - Uloží se výsledek do registru, popř. se iniciuje zapsání výsledku do paměti.
 - Určí se nový obsah PC.

Obvodová realizace procesoru

- Při návrhu procesoru se vychází ze **specifikace**, která obsahuje požadavky na instrukční soubor, architekturu, šířku dat a adres, výkonnost, plochu na čipu, spotřebu atd.
- Ukážeme si obvodovou realizaci nejjednoduššího procesoru, cílem bude demonstrovat princip činnosti procesoru.
- Použijeme tzv. **střádačovou architekturu** (tj. všechny instrukce ALU budou pracovat se speciálním registrem – střádačem (akumulátorem))
- Procesor bude připojen k paměti, která bude obsahovat program i data.
- Procesor bude navržen tak, že vytvoříme
 - datovou cestu (reprezentuje tok dat v procesoru) a
 - řadič (který bude implementován jako FSM).
- Než se dostaneme k realizaci procesoru, připomeneme si základní logické obvody – stavební bloky procesoru.

RAM

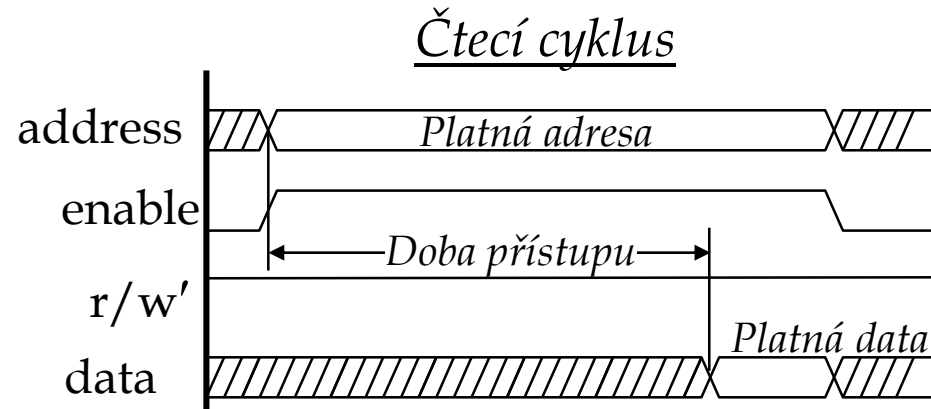


- RAM = *random access memory*
 - když $en = 1$ a $read/write = 1$, na datovou sběrnici *data* bude přivedena hodnota z paměti uložená na pozici, kterou definuje adresa vystavená na adresové sběrnici *address*
 - když $en = 1$ a $read/write = 0$, hodnota *data* přepíše data uložená v paměti na pozici, kterou definuje *address*
 - *en* značí signál *enable*, pokud $en = 0$, je paměť ve stavu vysoké impedance
 - nejjednodušší RAM jsou asynchronní
- SRAM vs DRAM
 - organizace, rychlost, cena, obnova (viz přednáška o pamětech)

Časování asynchronní RAM

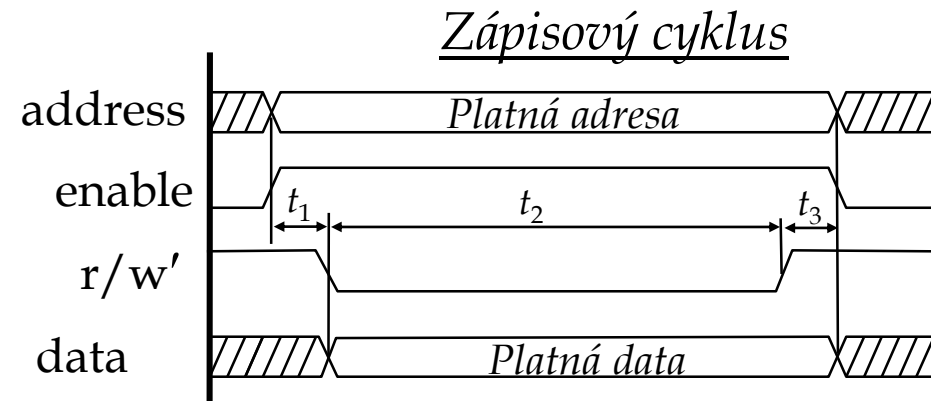
- *Čtecí cyklus*

- *Doba přístupu*: doba, za kterou získáme platná data od požádání (nastavení signálů a adresy)



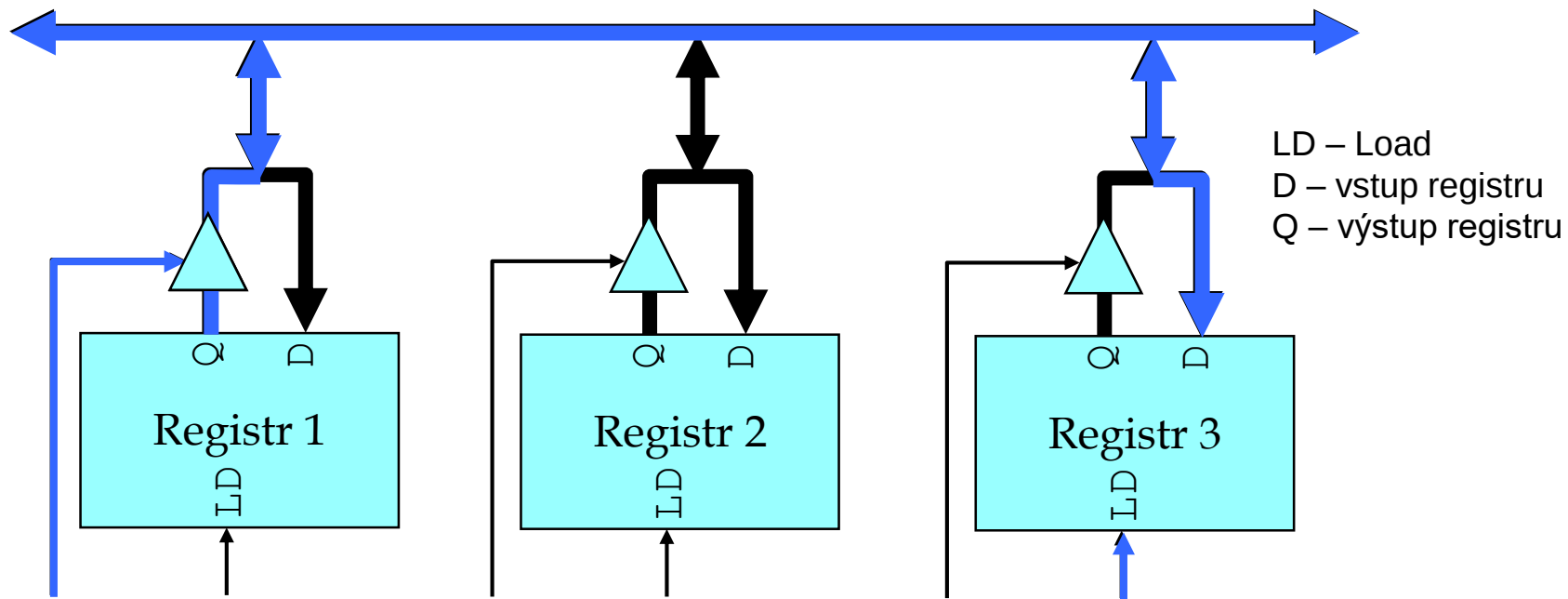
- *Zápisový cyklus*

- t_1 je min doba pro stabilizaci adresy před aktivací r/w
- t_2 je min doba, po kterou musí být vstupní data stabilní před deaktivací r/w
- t_3 je min doba, po kterou musí být adresa platná po deaktivaci r/w
- *Zápisový cyklus*: $t_1 + t_2 + t_3$

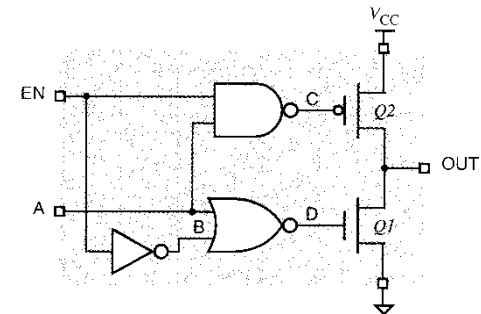


- Obvody, které používají RAM (tj. i náš procesor), musí správné časování zajistit!

Přenos dat s využitím sběrnice

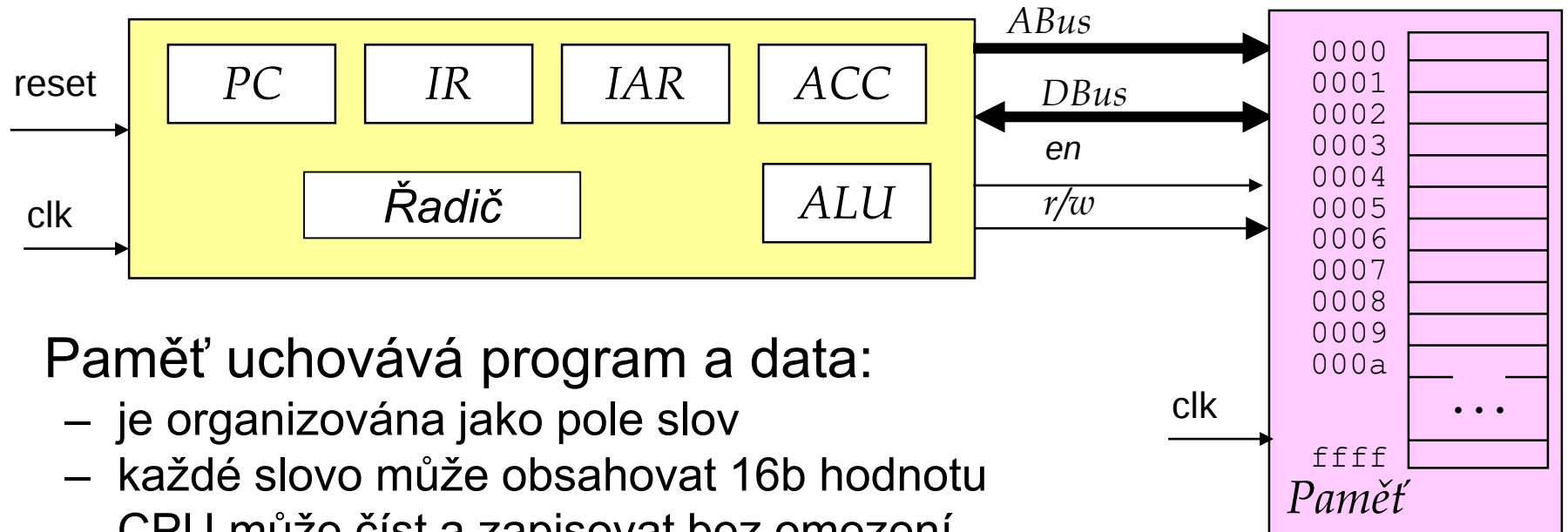


- **Sběrnice** je sdílená skupina vodičů sloužící pro přenos dat mezi několika zdroji/cíli
- Přenos dat zahrnuje:
 - umožnění jednomu zdroji dát data na sběrnici
 - nahrání dat do jednoho nebo několika cílových míst



Realizace 1b třístavového budiče

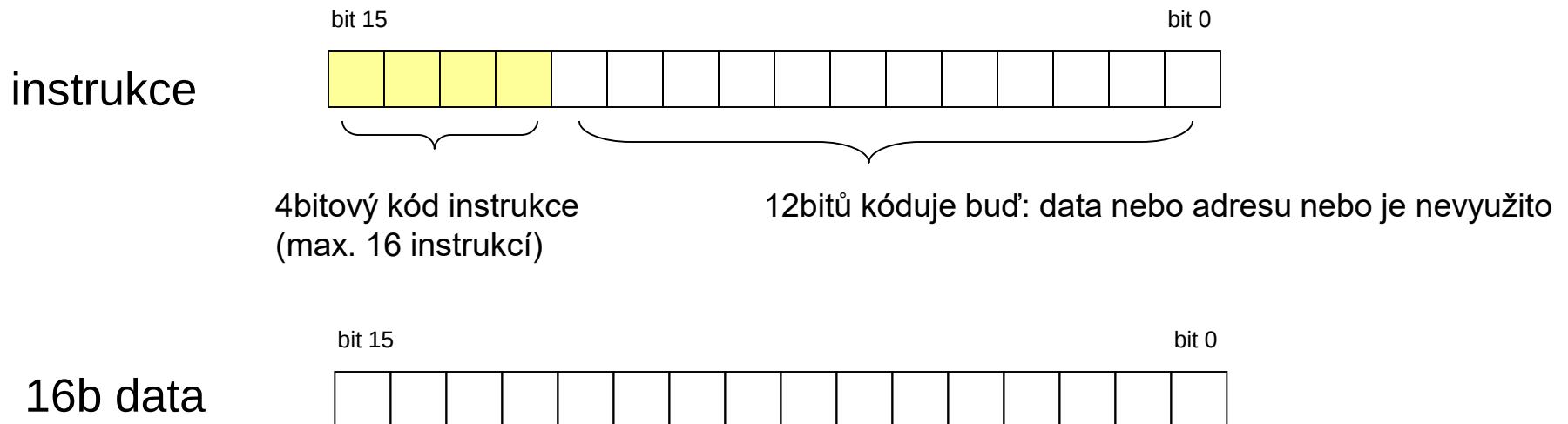
Nejjednodušší počítač



- Paměť uchovává program a data:
 - je organizována jako pole slov
 - každé slovo může obsahovat 16b hodnotu
 - CPU může číst a zapisovat bez omezení
- Zpracování instrukce
 - přečti slovo, jehož adresa je v *PC* (*Program Counter*) a inkrementuj *PC*
 - interpretuj tuto hodnotu jako instrukci (dekódování)
 - vykonej instrukci s využitím akumulátoru (*ACC*) a *ALU*
 - *IAR* (*Indirect Address Register*) – registr využívaný při nepřímém adresování

Instrukce procesoru

- Existuje řada způsobů, jak zvolit instrukce, zakódovat instrukce a zakódovat data.
- Abychom si v našem případě situaci co nejvíce ulehčili, bude procesor pracovat s **16bitovými slovy**, které mohou obsahovat současně jak kód instrukce, tak i data.



Sada instrukcí (1)

Hexa kód:

- 0000** zastav provádění programu (**halt**)
- 0001** vytvoř dvojkový doplněk z ACC (**negate**)
 $ACC := -ACC$
- 1xxx** nahrej do ACC hodnotu **xxx** (**mload**)
pokud je znaménkový bit xxx nulový, potom
 $ACC := 0xxx$ jinak $ACC := fxxx$
- 2xxx** nahrej do ACC hodnotu z adresy **xxx** (**dload**)
 $ACC := M[0xxx]$
- 3xxx** nahrej do ACC hodnotu, která je uložena na adrese, kterou definuje obsah buňky **xxx** (**iload**)
 $ACC := M[M[0xxx]]$
- 4xxx** ulož hodnotu z ACC na adresu **xxx** (**dstore**)
 $M[0xxx] := ACC$

Sada instrukcí (2)

Hexa kód:

- 5xxx** ulož hodnotu z ACC na adresu, která je určena hodnotou paměťové buňky s adresou xxx (**istore**)
 $M[M[0xxx]] := ACC$
- 6xxx** změň PC na xxx (**Branch**)
 $PC := 0xxx$
- 7xxx** změň PC na xxx jestliže $ACC = 0$ (**BrZero**)
if $ACC = 0$ then $PC := 0xxx$
- 8xxx** změň PC na xxx jestliže $ACC > 0$ (**BrPos**)
if $ACC > 0$ then $PC := 0xxx$
- 9xxx** změň PC na xxx jestliže $ACC < 0$ (**BrNeg**)
if $ACC < 0$ then $PC := 0xxx$
- axxx** přičti k ACC obsah paměťové buňky na adrese xxx (**Add**)
 $ACC := ACC + M[0xxx]$

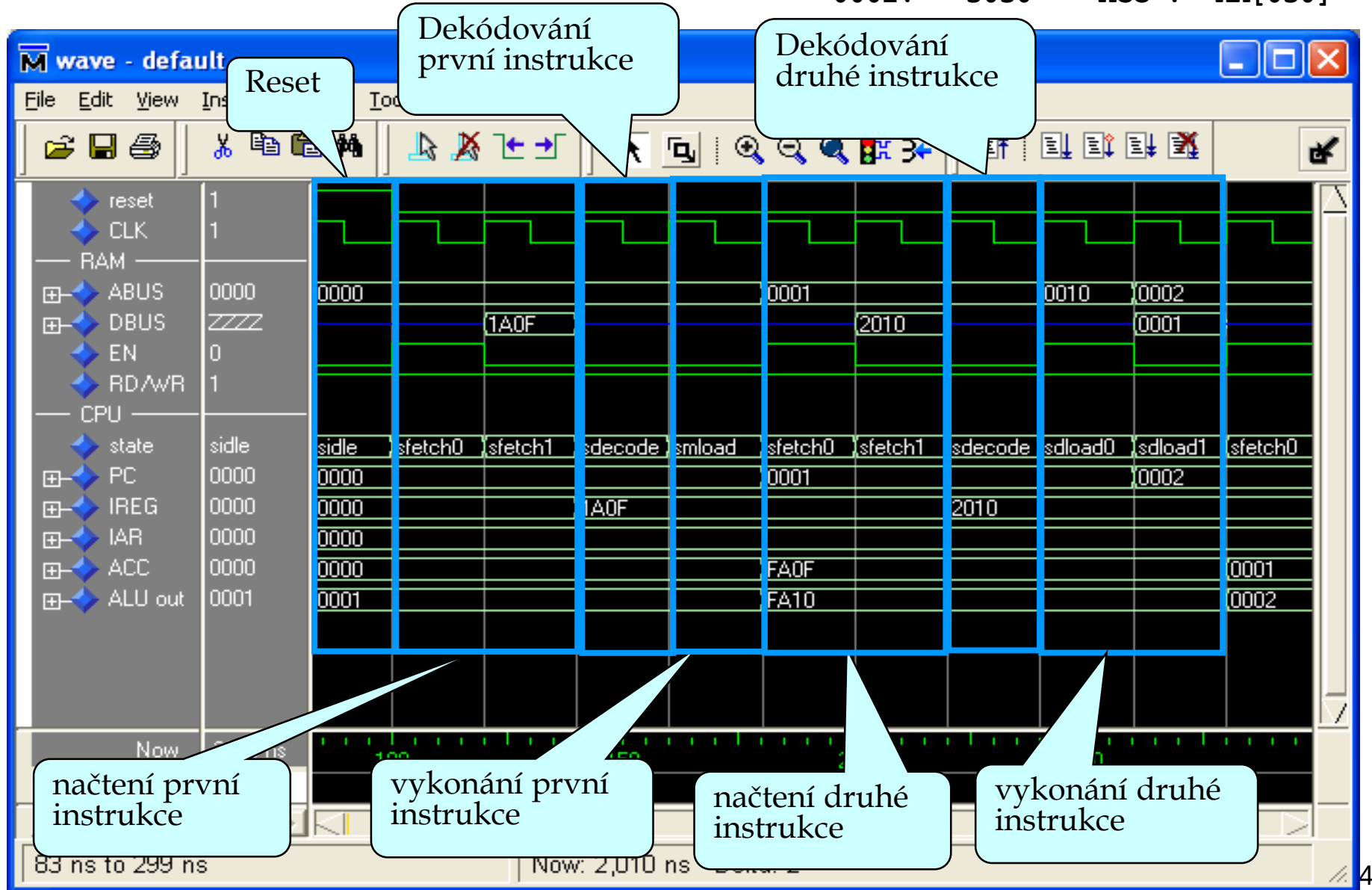
Příklad programu

Sečti hodnoty uložené na adresách 20-2f a zapiš výsledek na adresu 10.

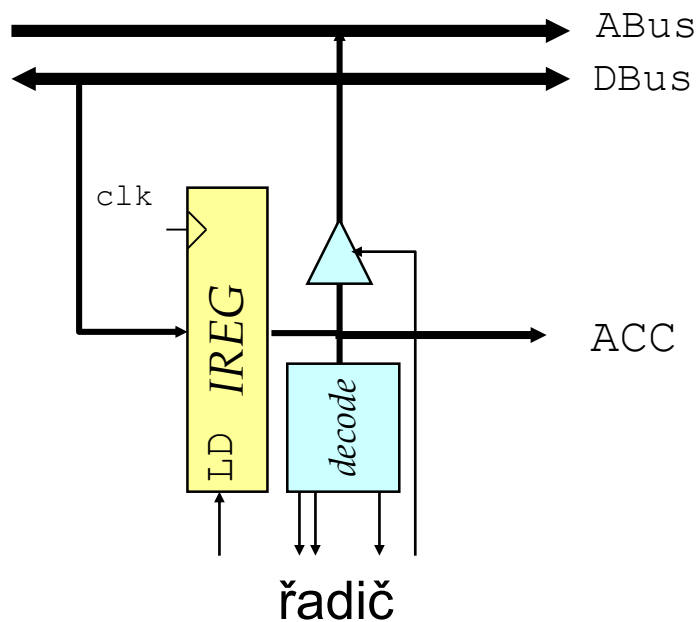
<i>Adresa</i>	<i>Instrukce</i>	<i>Komentář</i>
0000 (start)	1000 (ACC := 0000)	vynuluj součet S
0001	4010 (M[0010] := ACC)	
0002	1020 (ACC := 0020)	inicializuj pointer P
0003	4011 (M[0011] := ACC)	
0004 (loop)	1030 (ACC := 0030)	konec, pokud je P=030
0005	0001 (ACC := -ACC)	
0006	a011 (ACC := ACC + M[0011])	
0007	700f (if 0 goto 000f)	
0008	3011 (ACC := M[M[0011]])	S = S + *P
0009	a010 (ACC := ACC + M[0010])	
000a	4010 (M[0010] := ACC)	
000b	1001 (ACC := 0001)	P = P + 1
000c	a011 (ACC := ACC + M[0011])	
000d	4011 (M[0011] := ACC)	
000e	6004 (goto 0004)	goto loop
000f (end)	0000 (halt)	halt
0010		suma S
0011		pointer P

Př. časování instrukcí

Adresa	Hodnota	Význam
0000:	1A0F	ACC := FA0F
0001:	2010	ACC := M[010]
0002:	3030	ACC := MM[030]

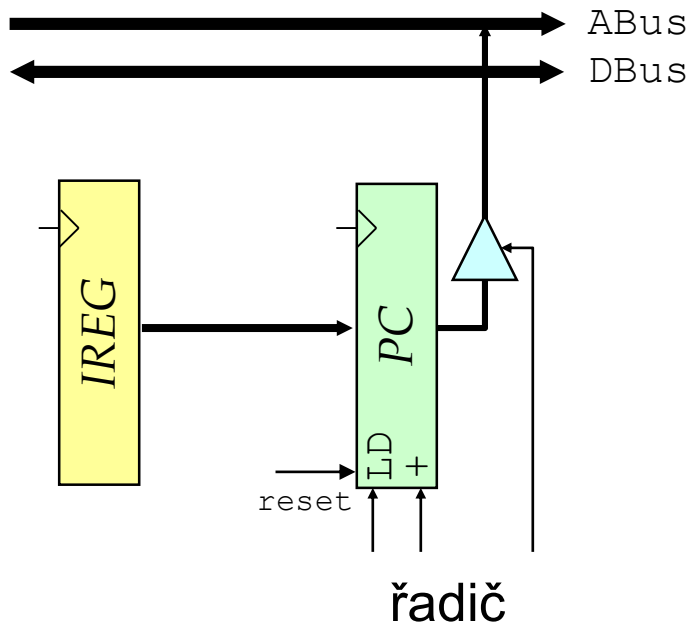


IREG a dekódování instrukce



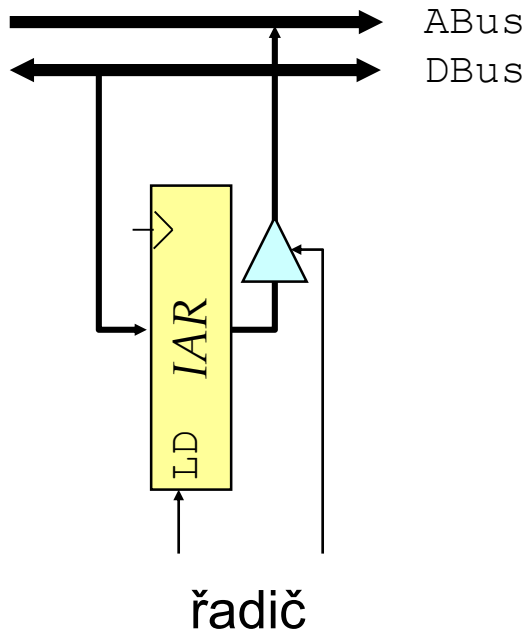
- Řadič vygeneruje signál LD a instrukce, která je na datové sběrnici, bude uložena do IREG.
- Pokud je instrukce v IREG, pak kombinační obvod *decode* nastaví na svém výstupu ty signály pro řadič, které budou v následujících taktech potřeba k vykonání dané instrukce. Např. pokud se jedná o instrukci ADD, potom se nastaví správná operace ALU, nastaví se datová cesta pro operandy apod.
- Vzhledem k tomu, že 16bitová slova přečtená z paměti obsahují jak instrukční kód tak i data, je třeba tato data přivést do střeďáče ACC.
- Při nepřímém adresování je součástí 16bitového slova, které je uloženo v IREG, i adresa paměti. Tuto adresu je třeba rovněž zpřístupnit na ABus. Za tímto účelem je vložen do schématu třístavový budič, který je ovládán řadičem. Budič je aktivován jen v tom případě, kdy je třeba dát adresu na ABus, jinak je jeho výstup ve stavu vysoké impedance.

Programový čítač

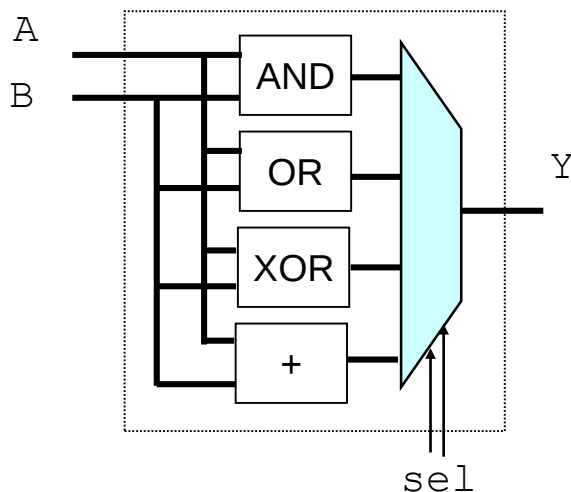
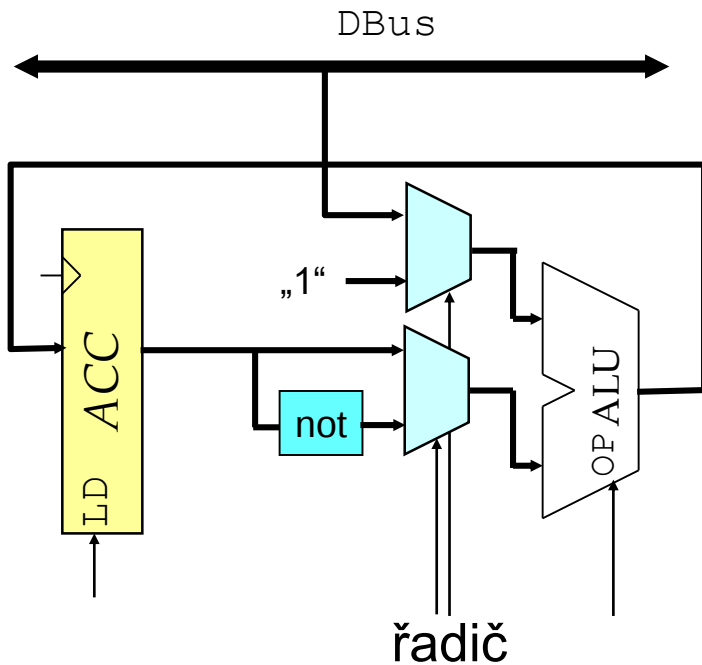


- Při resetu je nastavena počáteční hodnota PC = 0.
- Pokud se nejedná o skok, řadič generuje signál (+), který zvýší hodnotu PC o 1.
- Pokud se jedná o skok, je nová adresa uložena v IREG. Řadič generuje signál LD a načte tuto adresu do PC.
- Řadič zajistí vystavení hodnoty PC na ABus tak, že aktivuje řízení třístavového budiče.

Nepřímé adresování



- Při využití nepřímé adresace je na datovou sběrnici načtena adresa A, se kterou se bude následně pracovat.
- Adresa A je uložena do registru IAR tak, že řadič aktivuje signál LD.
- Pokud je následně třeba adresovat paměťovou buňku s adresou A, aktivuje řadič třístavový budič a tím je adresa A k dispozici na adresové sběrnici.

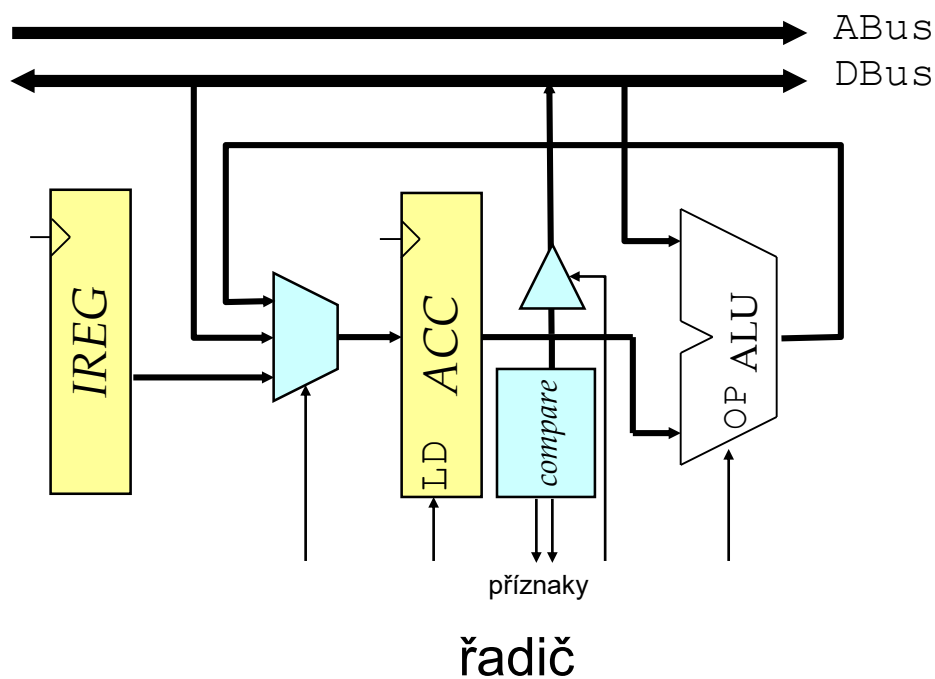


Princip realizace ALU

ALU

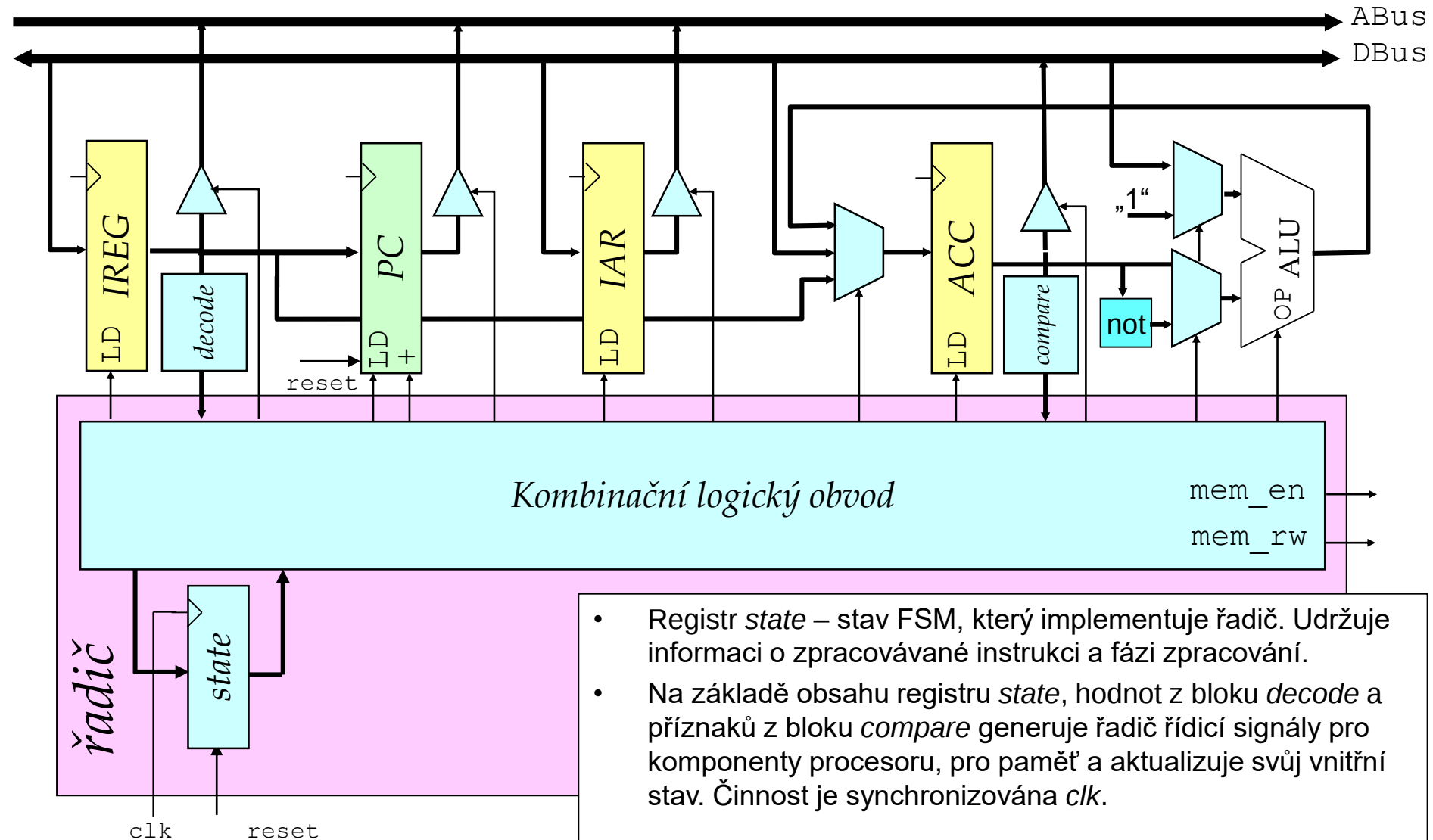
- ALU má dva operandy – střádač a hodnotu přečtenou z datové sběrnice.
- Výstup ALU je uložen do střádače, pokud řadič vygeneruje signál LD.
- Volba funkce ALU se děje pomocí signálu OP, který generuje řadič. Procesor zatím podporuje pouze instrukce sčítání a negace, ale není obtížné repertoár funkcí rozšířit.
- Pokud pracujeme v doplňkovém kódu a potřebujeme vytvořit zápornou hodnotu k obsahu střádače x , potom řadič vygeneruje signál pro multiplexory tak, aby došlo k sečtení negace x a „1“.

Střádač ACC



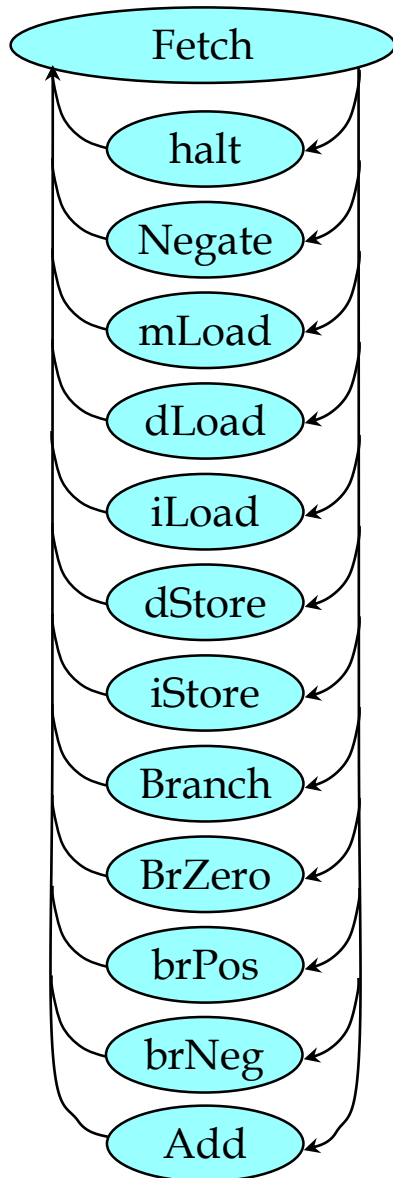
- Do střádače může být uložen buď výsledek operace ALU, hodnota dostupná na datové sběrnici nebo data uložená v IREG. Výběr provádí řadič pomocí řízení multiplexoru.
- Blok *compare* je kombinační obvod, který detekuje speciální stavy ACC – např. uloženou nulu, kladnou nebo zápornou hodnotu. Vytváří tak příznaky, na jejichž základě se rozhoduje řadič.
- Výstup ACC je možné zpřístupnit na datovou sběrnici (např. při instrukci store). Řadič musí aktivovat příslušný třístavový budič.
- Na obrázku nejsou nakresleny multiplexory na vstupu ALU, viz předchozí slide.

Celkové schéma procesoru



- Registr *state* – stav FSM, který implementuje řadič. Udržuje informaci o zpracovávané instrukci a fázi zpracování.
- Na základě obsahu registru *state*, hodnot z bloku *decode* a příznaků z bloku *compare* generuje řadič řídicí signály pro komponenty procesoru, pro paměť a aktualizuje svůj vnitřní stav. Činnost je synchronizována *clk*.

Instrukční cyklus

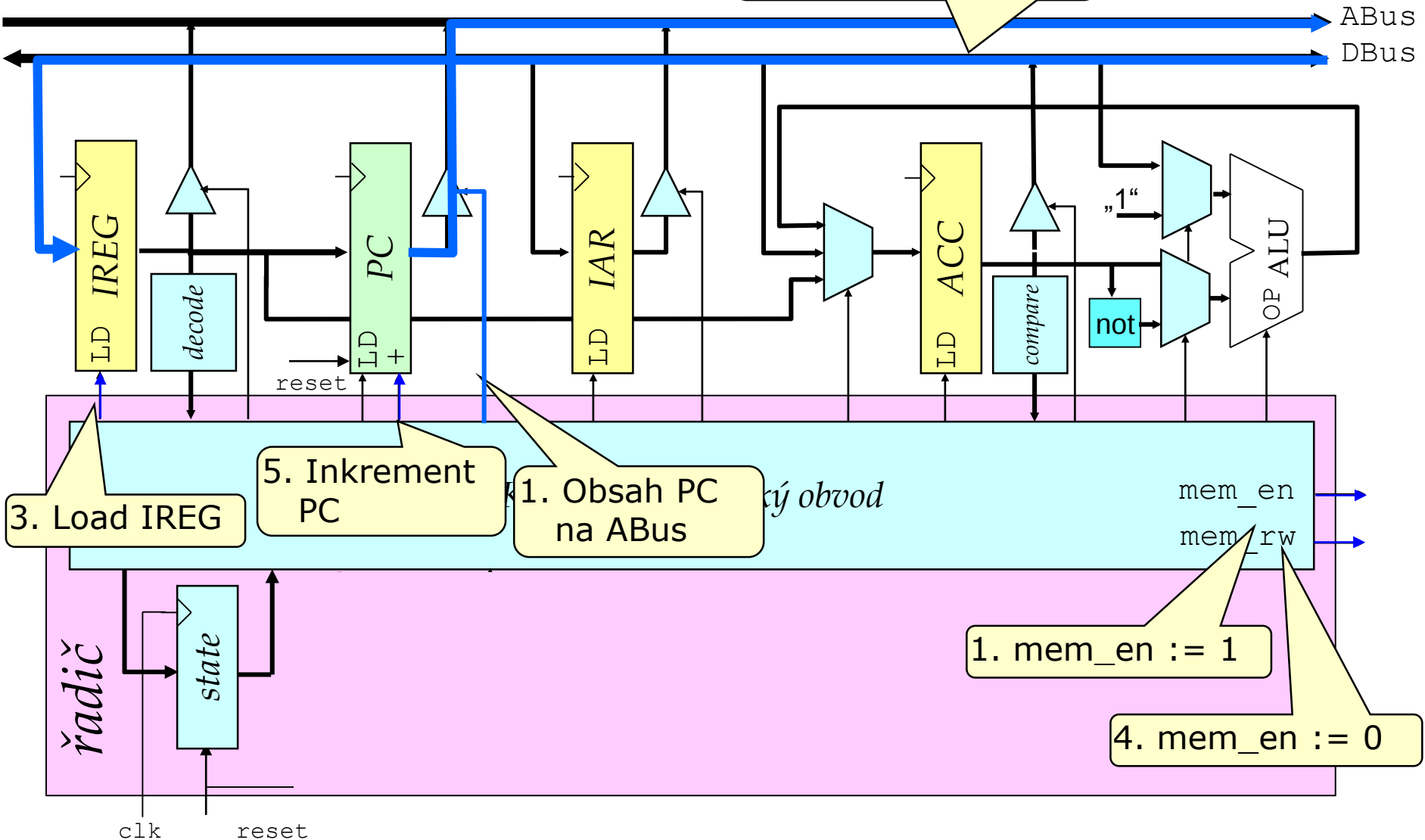


- **Načtení instrukce** (Instruction fetch)
 - Dle obsahu *PC* je přečtena instrukce z paměti
 - *PC* je inkrementován
- **Dekódování instrukce** (Instruction decode)
 - podle nejvyšších 4 bitů se určí, co se bude dělat
 - aktivují se příslušné obvody
- **Provedení instrukce** (Instruction execution)
 - načtení dalších potřebných dat
 - vykonání instrukce
 - zápis do paměti
 - modifikace *PC*, *ACC* atd.
 - může trvat pro různé instrukce různě dlouho

Vykonání instrukce - příklady

- Přímé čtení
 - přenes data z paměti do ACC, využij 12 nižších bitů jako adresu
 - vyžaduje generování signálů pro paměť a ACC
- Podmíněný skok
 - otestuj zda $ACC=0$ (nebo >0 nebo <0)
 - pokud ano, přenes nižších 12 bitů instrukčního slova do PC
- Nepřímý zápis
 - přenes data z paměti do IAR (Indirect Address Register) s využitím nižších 12 bitů instrukčního slova jako adresy
 - přenes data z ACC do paměti s využitím obsahu *IAR* jako adresy
 - vyžaduje zaslání hodnoty z *IAR* na adresovou sběrnici a nastavení signálů pro zápis do paměti

Fetch – načítání instrukce

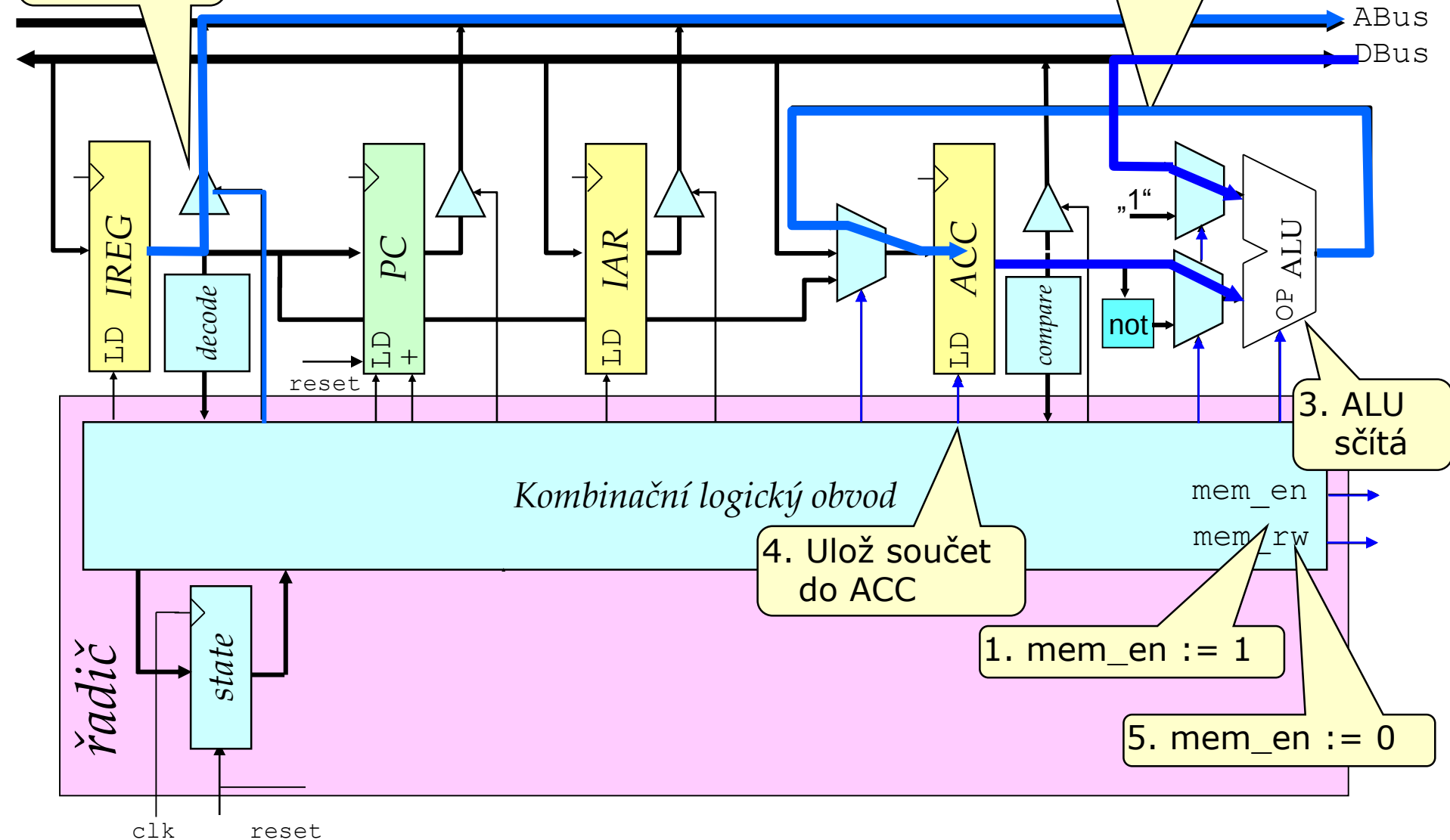


Provedení instrukce ADD

$ACC := ACC + M[0xxx]$

1. IREG na ABus

2. Obsah paměti na DBus, data z ACC



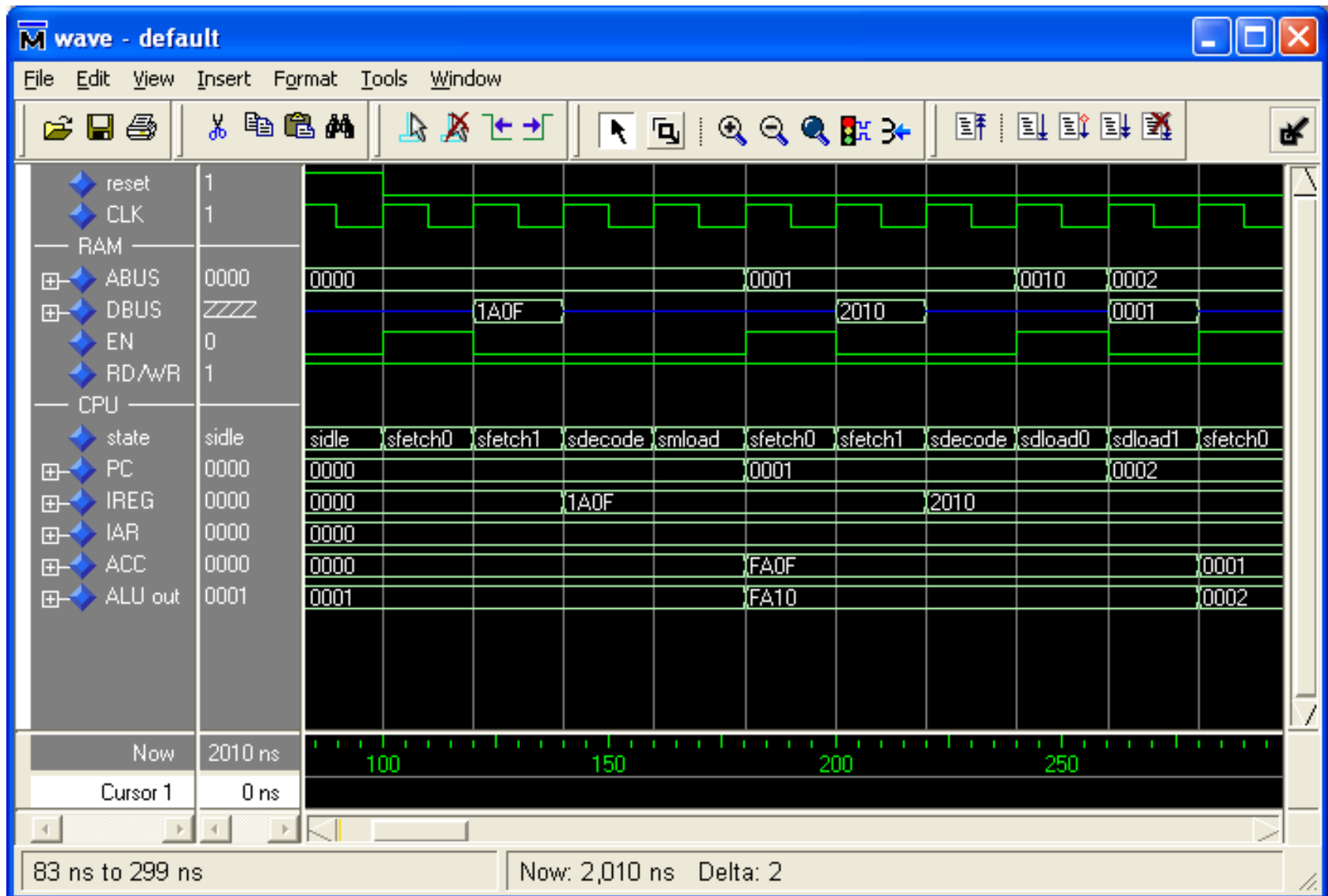
Testovací program

```
ram(0)  <= x"1a0f"; -- immediate load
ram(1)  <= x"2010"; -- direct load
ram(2)  <= x"3030"; -- indirect load
ram(3)  <= x"4034"; -- direct store
ram(4)  <= x"0001"; -- negate
ram(5)  <= x"2034"; -- direct load
ram(6)  <= x"0001"; -- negate
ram(7)  <= x"5032"; -- indirect store
ram(8)  <= x"0001"; -- negate
ram(9)  <= x"1fff"; -- immediate load
ram(10) <= x"a008"; -- add
ram(11) <= x"700d"; -- brZero
ram(12) <= x"0000"; -- halt
ram(13) <= x"1400"; -- immediate load
ram(14) <= x"8010"; -- brPos
ram(15) <= x"0000"; -- halt
ram(16) <= x"0001"; -- negate
ram(17) <= x"9013"; -- brNeg
ram(18) <= x"0000"; -- halt
ram(19) <= x"6015"; -- branch

ram(20) <= x"0000"; -- halt
ram(21) <= x"8014"; -- brPos
ram(22) <= x"7014"; -- brZero
ram(23) <= x"0001"; -- negate
ram(24) <= x"9014"; -- brNeg
ram(25) <= x"0000"; -- halt
ram(48) <= x"0031"; -- pointer for iload
ram(49) <= x"5af0"; -- target of iload
ram(50) <= x"0033"; -- pointer for istore
ram(51) <= x"0000"; -- target of istore
ram(52) <= x"f5af"; -- target of dstore
```

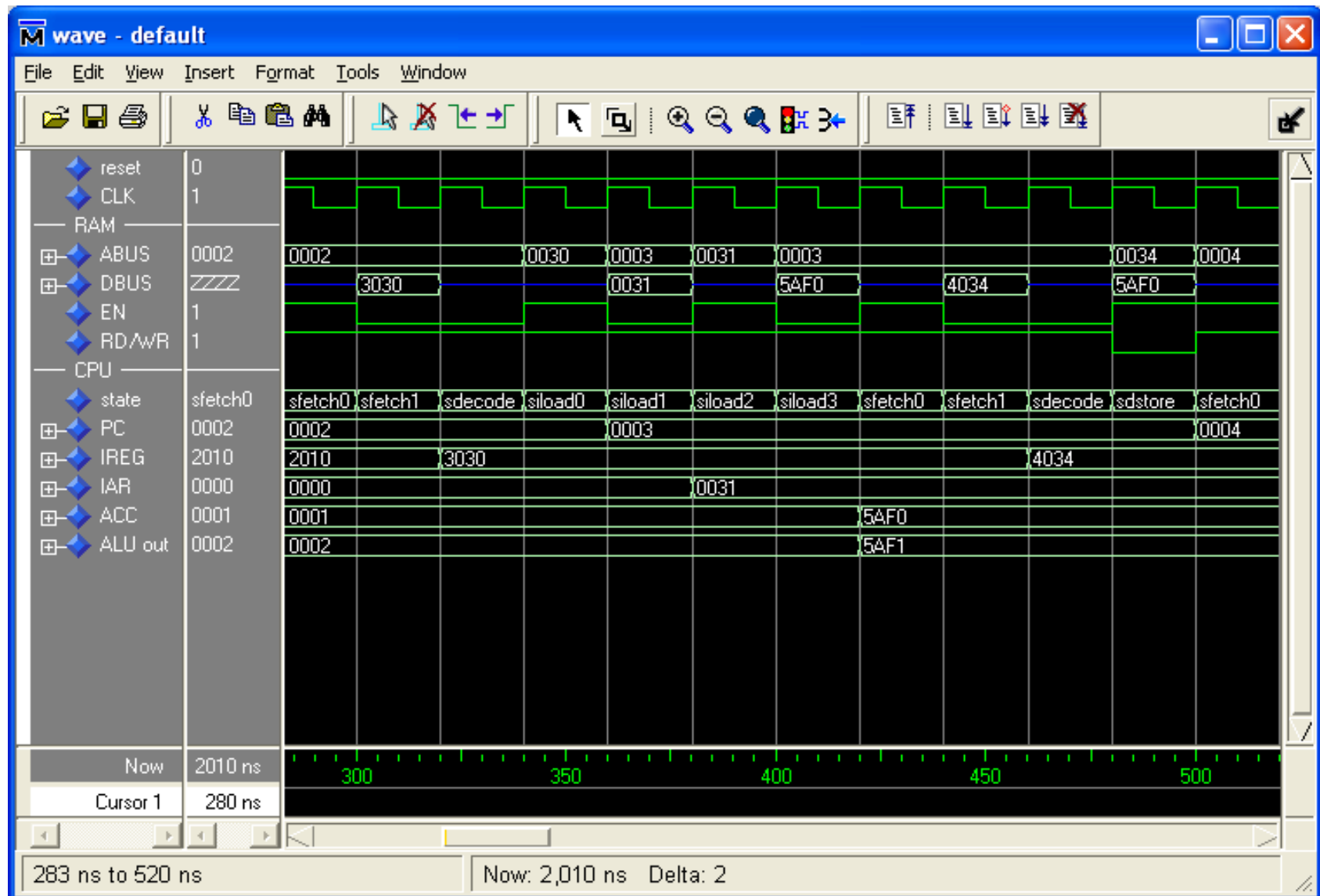
Simulace procesoru (1)

1a0f -- immediate load, 2010 -- direct load



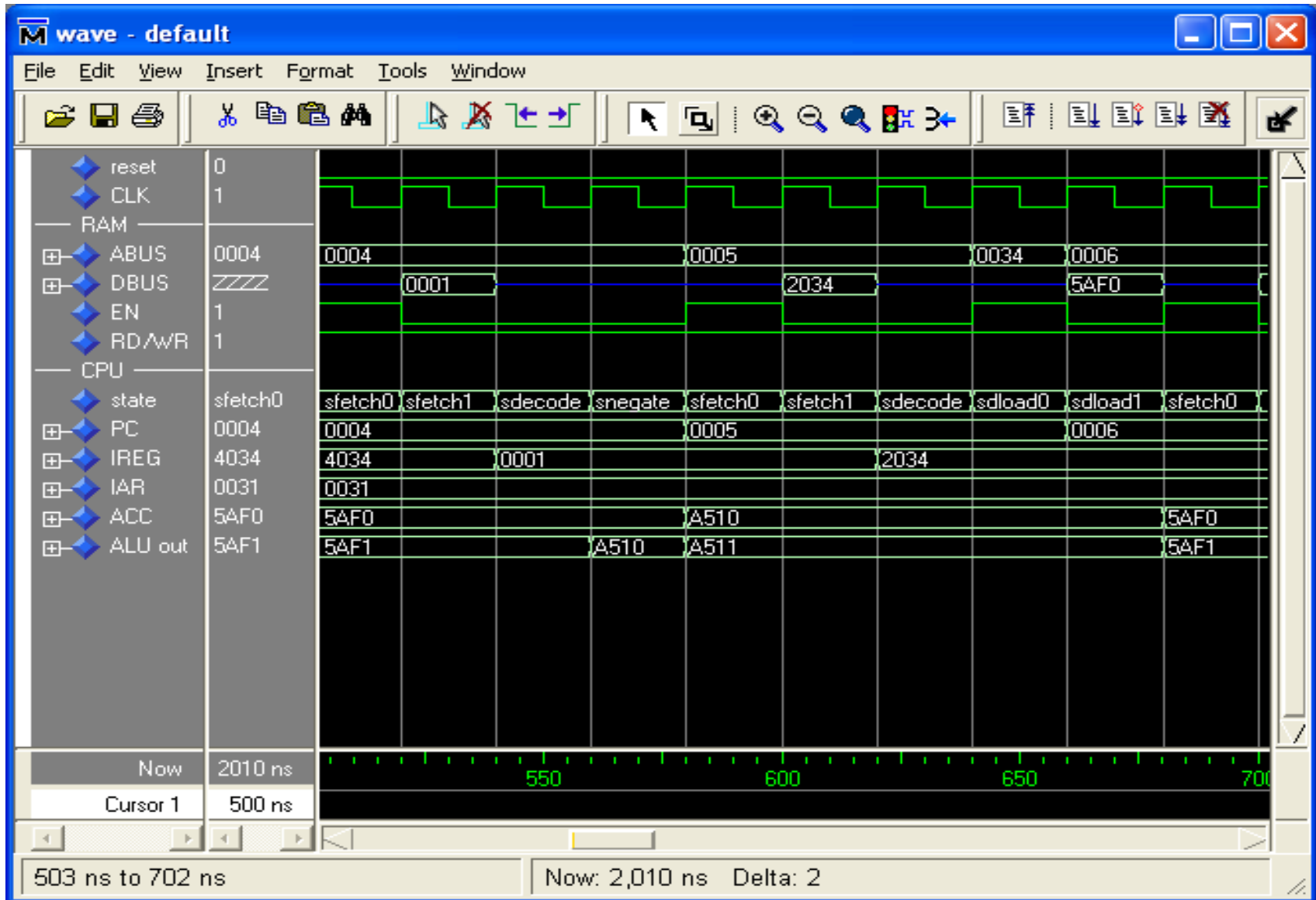
Simulace procesoru (2)

3030 -- indirect load, 4034 -- direct store



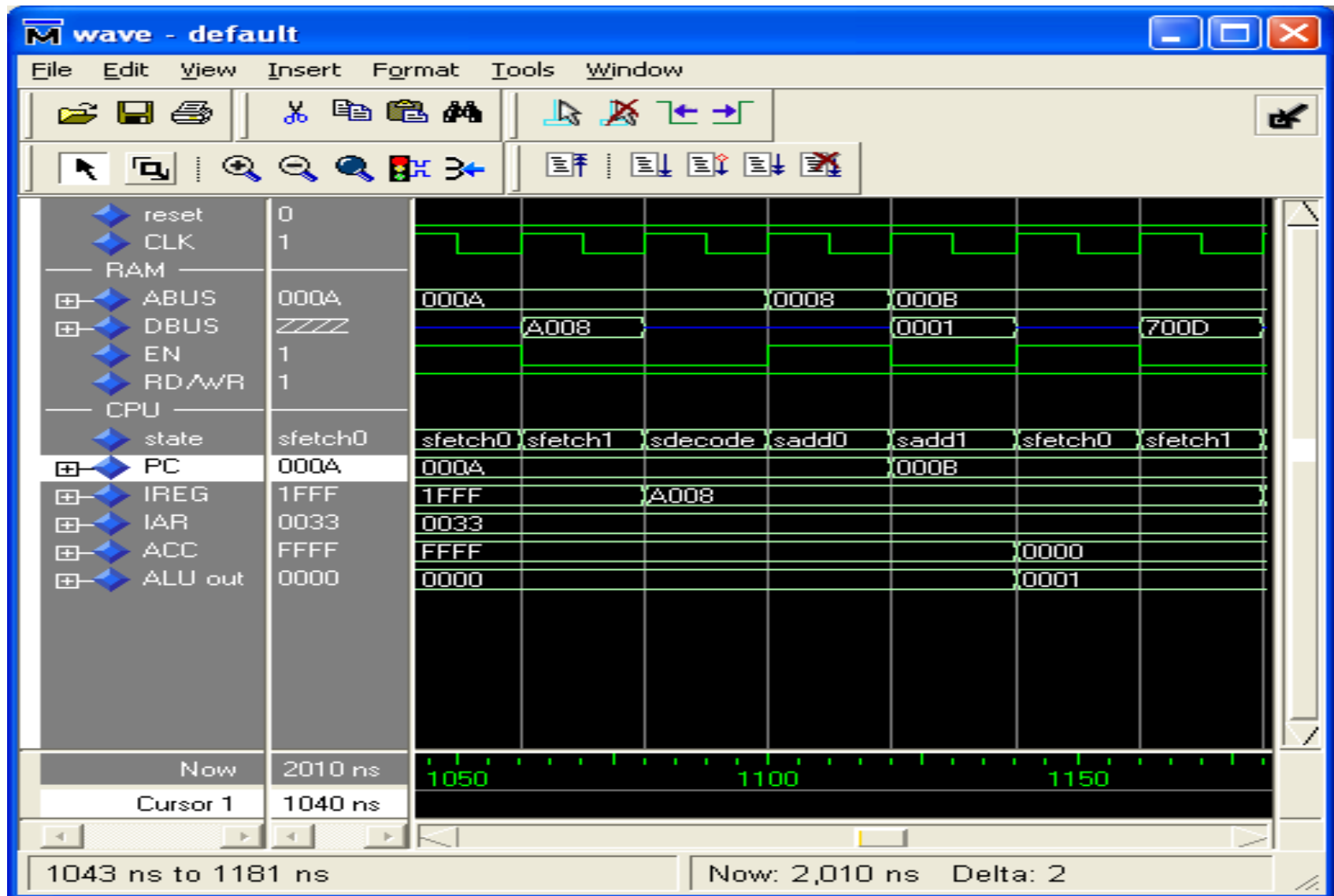
Simulace procesoru (3)

0001 – negate, 2034 -- direct load

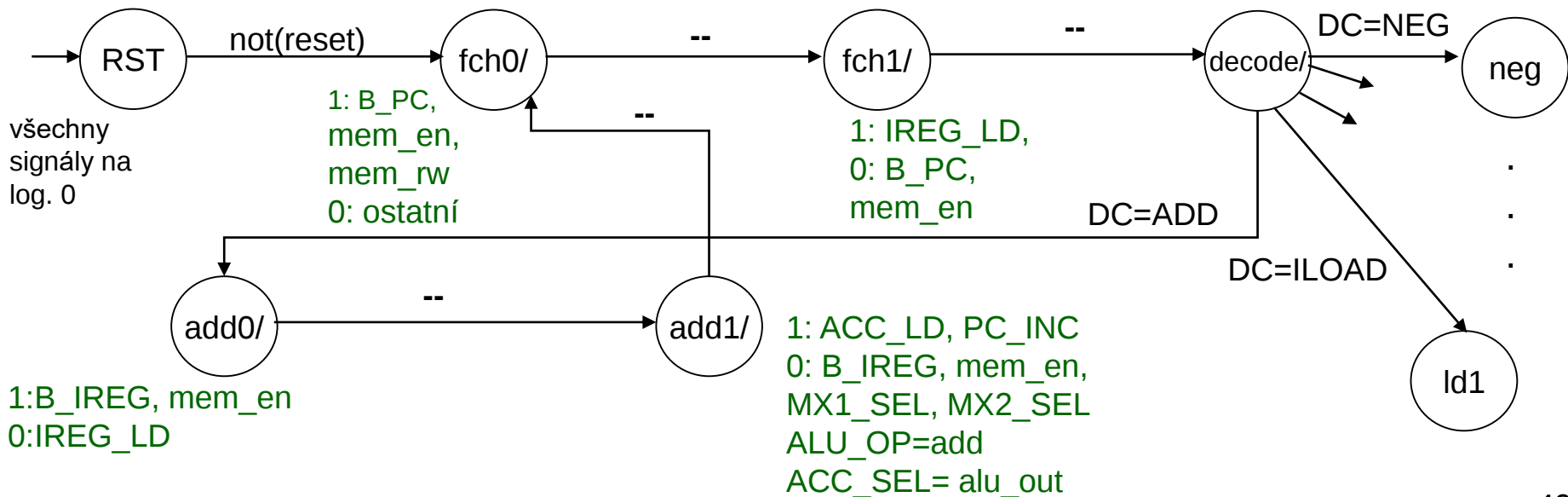
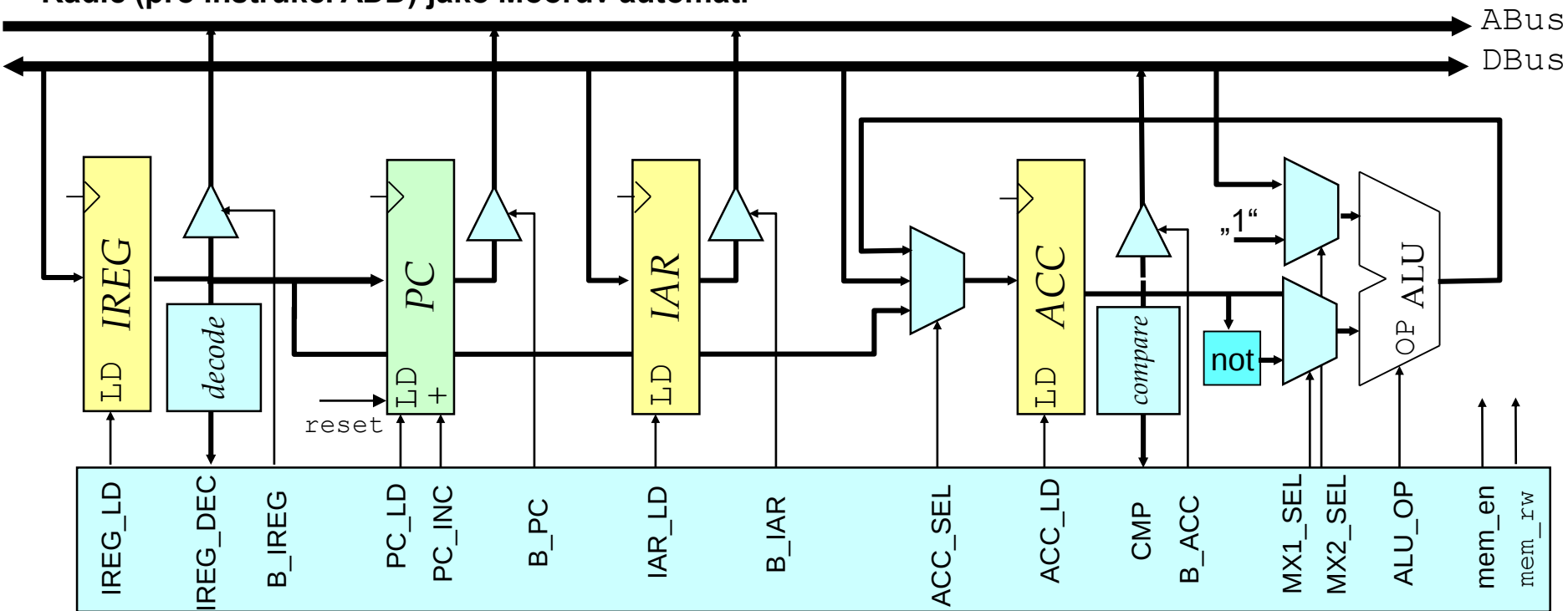


Simulace procesoru (4)

a008 – add ... atd.



Řadič (pro instrukci ADD) jako Moorův automat.



Řadič – poznámky k obrázku

- Implementován jako Moorův automat (signály, které se (de)aktivují v příslušném stavu, jsou uvedeny vedle každého stavu).
 - Fetch – 2 stavy
 - Decode – 1stav
 - ADD – 2 stavy
 - Další instrukce: 1-4 cykly (graf příslušných automatů neuveden, k přechodu dojde na základě hodnoty signálu DC)
- Pokud je aktivován signál reset, přejde se z libovolného stavu do stavu RST (příslušné hrany nejsou uvedeny).
- Značka „--“ označuje přechod, který se vždy vykoná.

Zdokonalování von Neumannovy koncepce

- Indexregistry (1949), EDSAC (U. of Cambridge)
- Jednotka pro operace s pohyblivou řádovou čárkou FPU (Floating Point Unit, 1954), IBM 704, NORC
- Přerušování programu (1956), UNIVAC 1103, Remington Rand
- Univerzální registry (1956), PEGASUS, Ferranti
- Asynchronní činnost vstup-výstupních periferních zařízení (1956), UNIVAC LARC, Remington Rand
- Nepřímé adresování (1958), IBM 709
- Virtuální paměť (1959), ATLAS, Univ. of Manchester
- V 60. letech již k zásadním zdokonalením architektury nedocházelo tak často, rozvoj se ubíral cestou zdokonalování technologie (následující slide), která vedla k zvětšování výkonnosti i kapacit pamětí.
- Za významné jevy 70. let považujeme integrovaný procesor (mikroprocesor) a použití rychlé vyrovnávací paměti cache, viz dále.
- V dalším období docházelo ke zvyšování úrovně paralelismu na různých úrovních: řetězené zpracování instrukcí, současné vydávání instrukcí, vícejádrové systémy ...
- S těmito a dalšími koncepty se v INP ještě setkáme.

Zdokonalování technologie: Moorův zákon (1965)

Moore's Law: The number of transistors on microchips doubles every two years

Our World
in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Transistor count

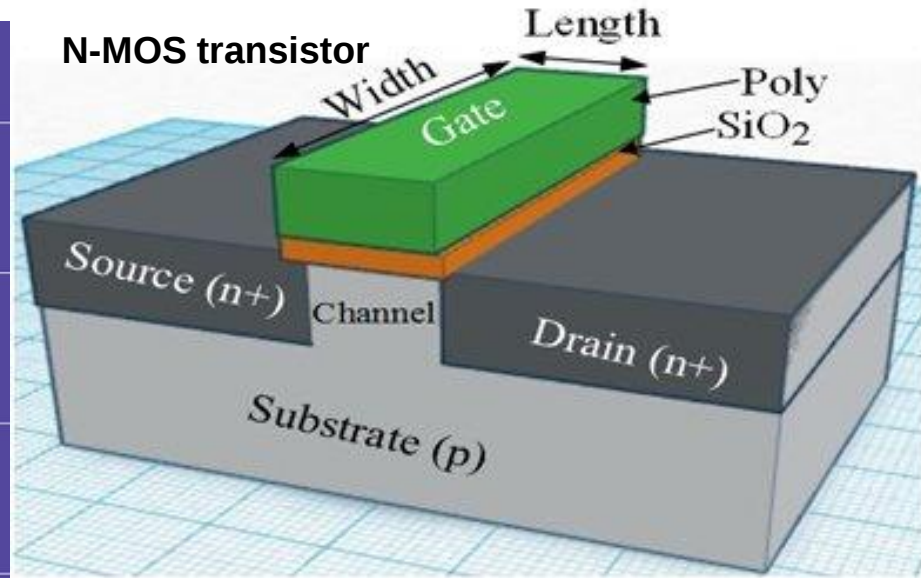
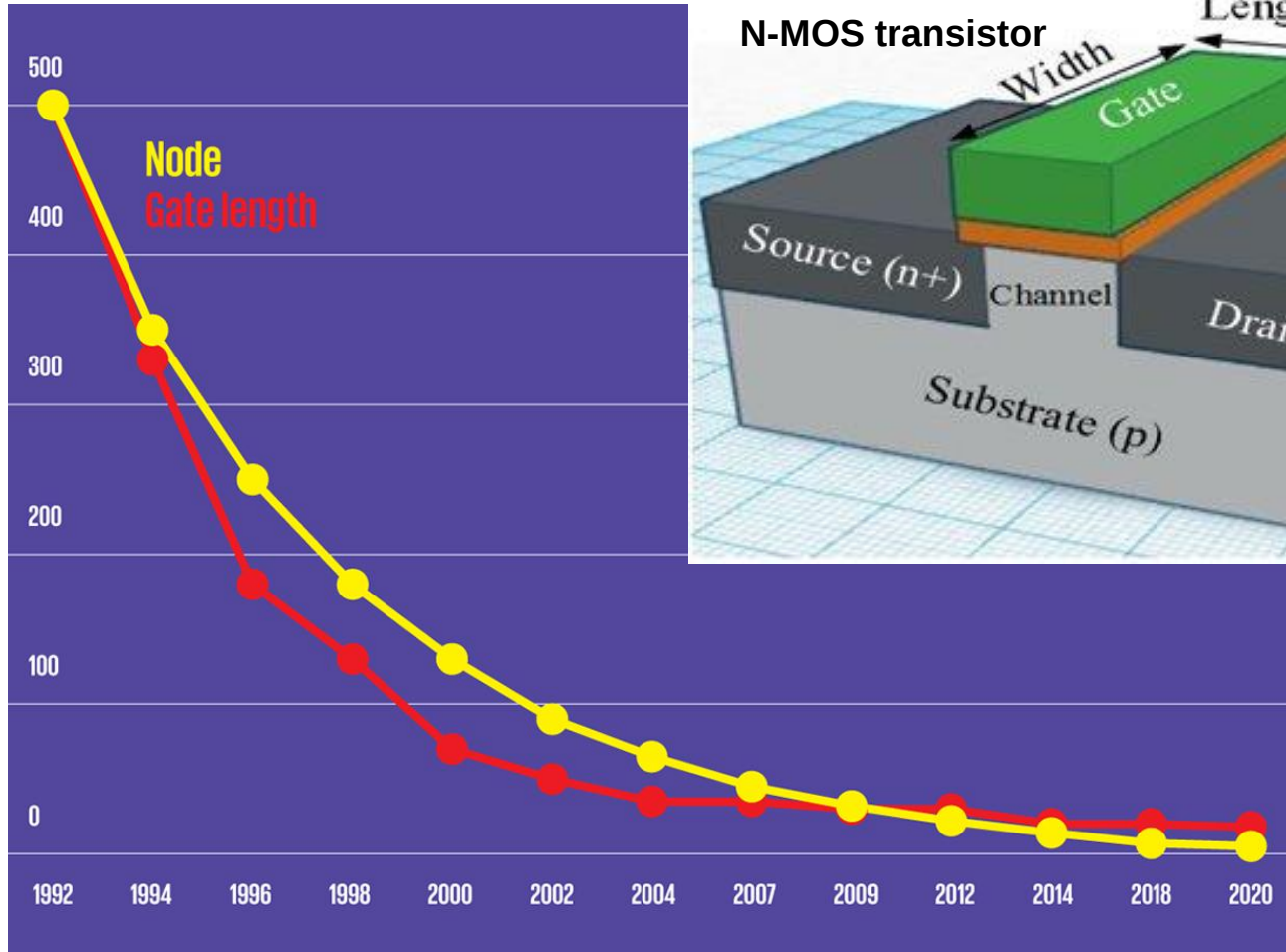


Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

OurWorldinData.org – Research and data to make progress against the world's largest problems.

Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

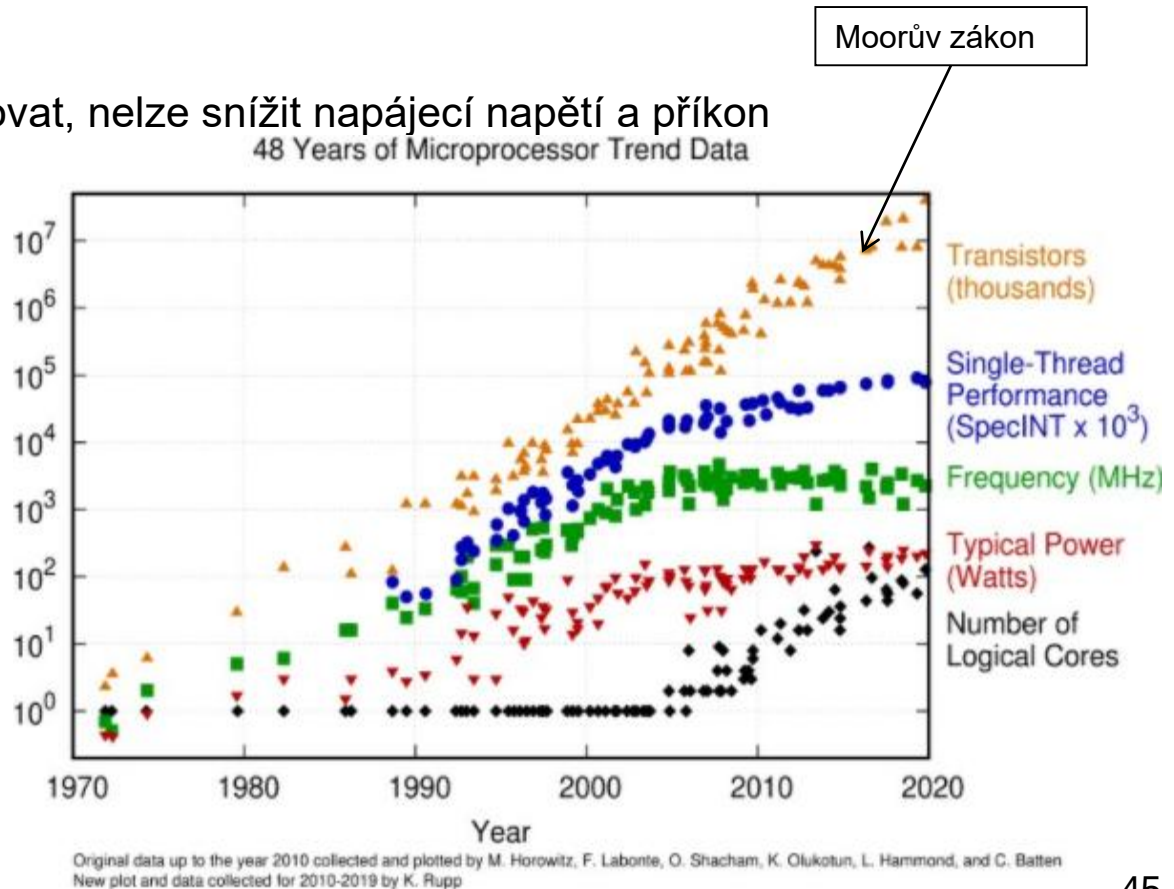
“Technology node” vs. “Gate Length”



S. K. Moore: A Better Way to Measure Progress in Semiconductors, IEEE Spectrum 7/2020
P. H. Vora, R. Lad: A Review Paper on CMOS, SOI and FinFET Technology
<https://www.design-reuse.com/articles/41330/cmos-soi-finfet-technology-review-paper.html>

Vývoj technologie

- *Dennard scaling*: Po desetiletí bylo možné **zvyšovat** hustotu integrace (zmenšovat tranzistory) a **zvyšovat** frekvenci. **Snížením** napájecího **napětí** byl zajištěn akceptovatelný příkon.
- Současnost:
 - frekvenci nelze moc zvyšovat, nelze snížit napájecí napětí a příkon
 - vícejádrové procesory běžící na nižším kmitočtu
 - inteligentní řízení příkonu
- Fenomén: **Dark silicon**
 - na čipu je mnoho tranzistorů, ale jen část může být současně využita, jinak čip shoří



Literatura

- Odkazy na slidech
- Drábek, V. Výstavba počítačů. Skriptum VUT, 1995
- Turner, J.: Simple processor, kurz CS/EE 260, Washington University in St. Louis, 2003