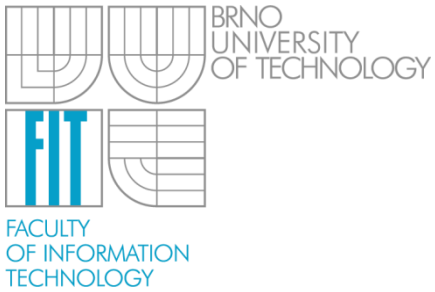
Návrh číslicových systémů (INC)

Otto Fučík

Vysoké učení technické v Brně
Fakulta informačních technologií
Božetěchova 2, 612 66 Brno



Použitá literatura

N. Frištacký, M. Kolesár, J. Kolenička a J. Hlavatý: „Logické systémy", SNTL Praha, 1986
M. Eysselt: „Logické systémy", SNTL Praha, skriptum VUT v Brně, 1985
J. F. Wakerly: „Digital Design. Principles and Practices", Prentice Hall, ISBN 0-13-769191-2, 2000
V. P. Nelson, H.T.Nagle, B.D.Carroll, J.D.Irwin: „Digital Logic Circuit Analysis & Design", ISBN 0-13-463894-8, 1995
T.L.Floyd: „Digital Fundamentals", Prentice Hall, ISBN 0-13-080850-4, 2000

Systémy

Implementace algoritmu

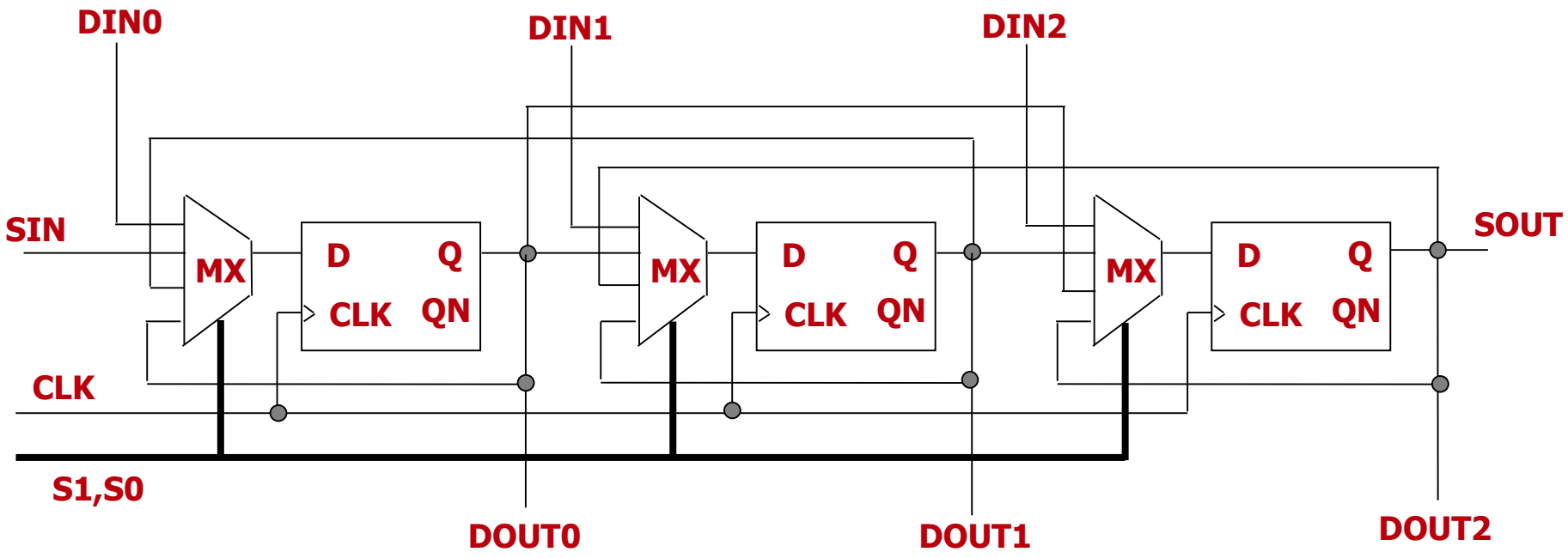
- Aplikačně specifické výpočetní systémy
  - Funkce je optimalizována pro danou (specifickou funkci)
  - Vysoká efektivita výpočtů (velká výkonnost, nízké energetické nároky)
  - Složitý návrh a výroba = vysoká cena (vyplatí se při hromadné výrobě)
  - Lze modifikovat (programovat, konfigurovat) jen v omezené míře
- Univerzální výpočetní systémy (počítače)
  - Určeny pro obecné použití (univerzální)
    - Amortizace návrhu a výroby výpočetního stroje
    - Mohou být programovány tak, aby řešily (potenciálně) libovolné (vyčíslitelné) úlohy = levný návrh aplikací (software)
  - Nižší efektivita výpočtů oproti aplikačně specifických systémům
  - Dnes jsou často vestavěny do větších systémů (anglicky Embedded Systems), kde vykonávají specifickou funkci

Implementace algoritmu

- Software (SW)
  - Výpočet běží na univerzálním procesoru
    - Datové struktury
    - Řídicí struktury – sekvence, rozhodování, iterace
- Hardware (HW)
  - Specializované obvody pro konkrétní úlohu
    - Datové struktury – registry, paměti apod.
    - Řídicí struktury – řadič řídí funkční jednotky a datovou cestu
  - Urychlení, nižší cena a spotřeba pro vhodnou třídu aplikací
    - Ne univerzální stroj, ale aplikačně specifický HW – implementuje se jen to, co je nezbytně třeba pro danou funkci
  - Realizace
    - Jako kombinační, nebo sekvenční obvod, či jejich kombinace – závisí na požadované výkonnosti (zpoždění), příkonu, kapacitě paměti, ceny, zkušeností, požadované rychlosti návrhu, dostupnosti návrhových prostředků atd.

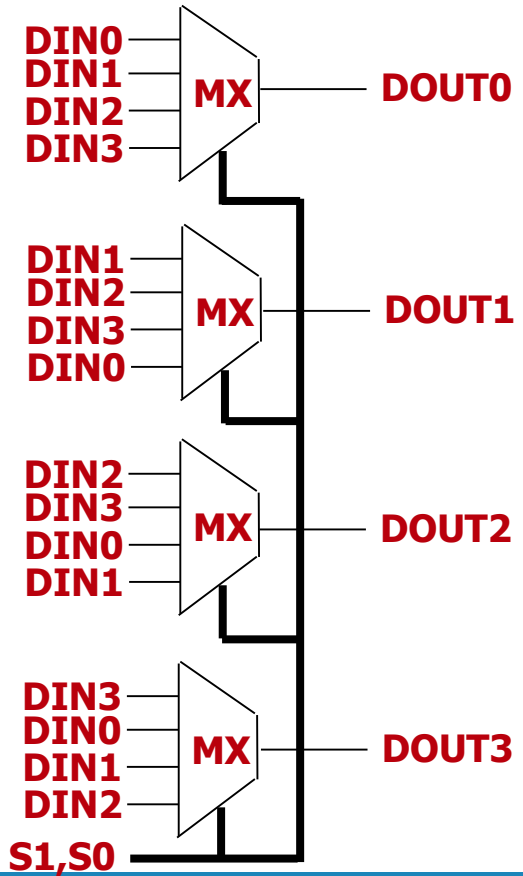
Implementace algoritmu: Sekvenční obvod

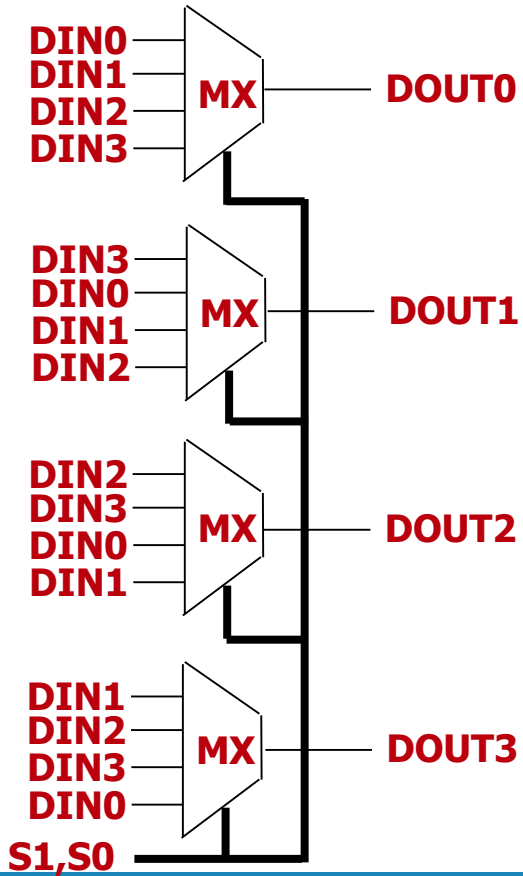
- Posuvný registr
  - DIN[2:0] – Paralel Data In
  - DOUT[2:0] – Paralel Data Out
  - SIN – Serial Data In
  - SOUT – Serial Data Out
  - S0,S1 – výběr funkce
- Posuv vlevo, vpravo, paralelní uložení informace a zapamatování stavu
  - S1=0, S0=0 – Paralel Load
  - S1=0, S0=1 – Shift Right
  - S1=1, S0=0 – Rotate Left
  - S1=1, S0=1– Hold



Implementace algoritmu: Kombinační obvod

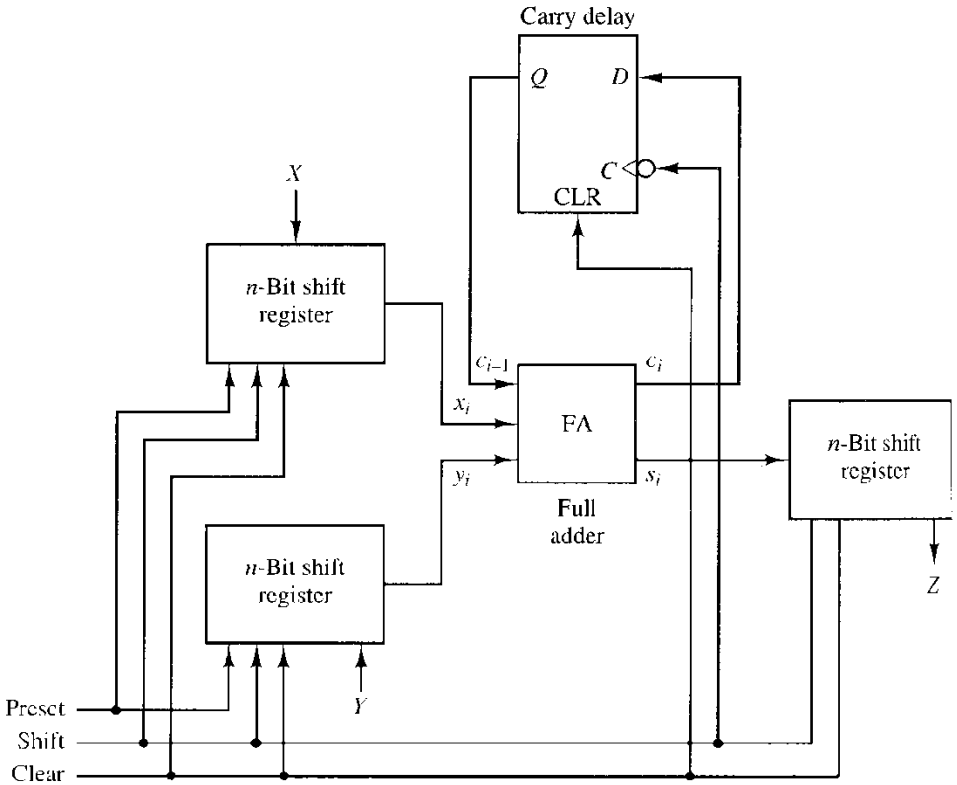
- Válcový posouvač (barrel shifter)
- Příklad: rotace vpravo v jednom kroku
  - S1=0, S0=0 – rotace o 0 bitů vpravo
  - S1=0, S0=1 – rotace o 1 bit vpravo
  - S1=1, S0=0 – rotace o 2 bity vpravo
  - S1=1, S0=1 – rotace o 3 bity vpravo
- Rotace vlevo v jednom kroku
  - S1=0, S0=0 – rotace o 0 bitů vlevo
  - S1=0, S0=1 – rotace o 1 bit vlevo
  - S1=1, S0=0 – rotace o 2 bity vlevo
  - S1=1, S0=1 – rotace o 3 bity vlevo



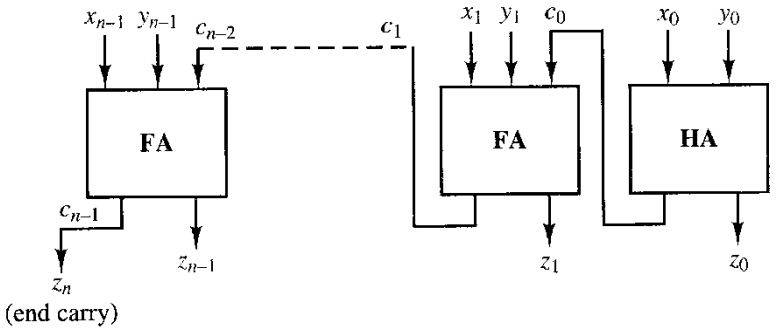


Implementace algoritmu: Řadič + datová cesta

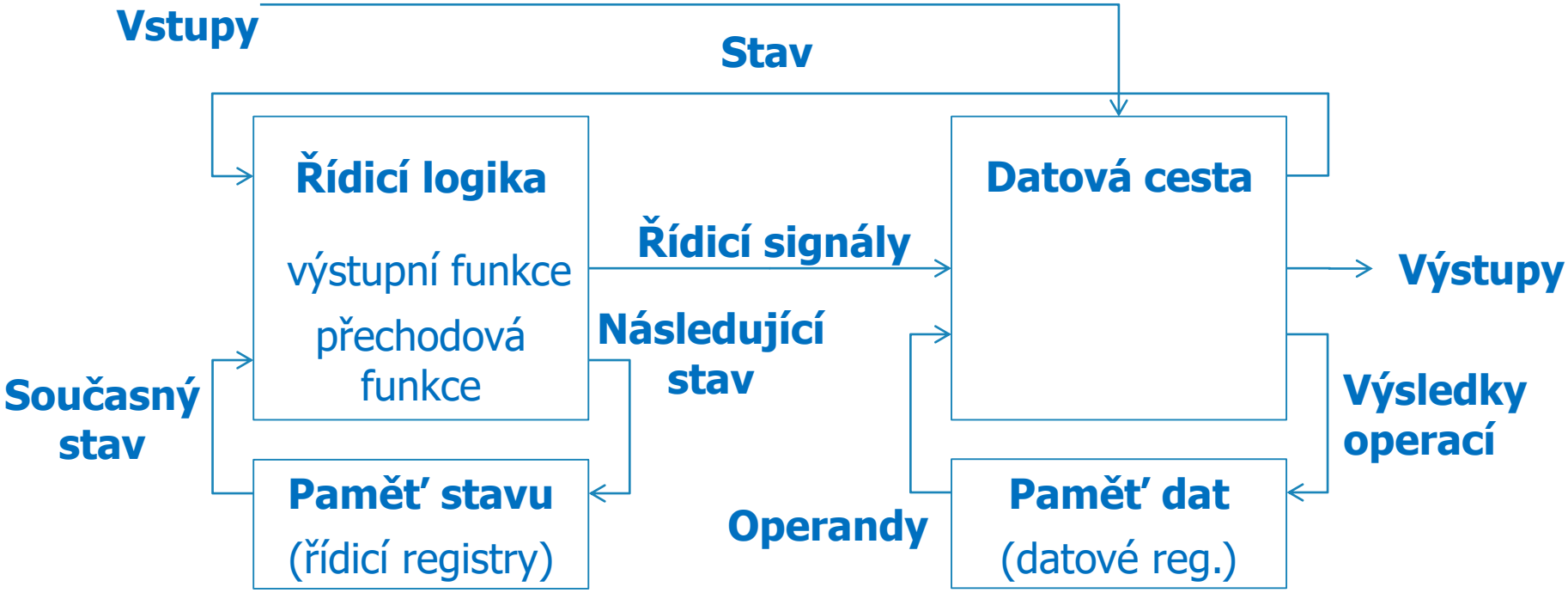
- Příklad - sériová sčítačka
  - Do posuvných registrů X a Y se paralelně zapíší hodnoty operandů
  - Vynuluje se registr
  - S každým taktem hodin se postupně vysouvají jednotlivé bity operandů od LSB k MSB a přivádí se na vstupy sčítačky
  - V registru se pamatují případné přenosy
  - Výstup sčítačky je přiveden na sériový vstup posuvného registru Z
  - Po n krocích je v Z výsledek



- Kombinační sčítačka



Implementace algoritmu: Řadič + datová cesta

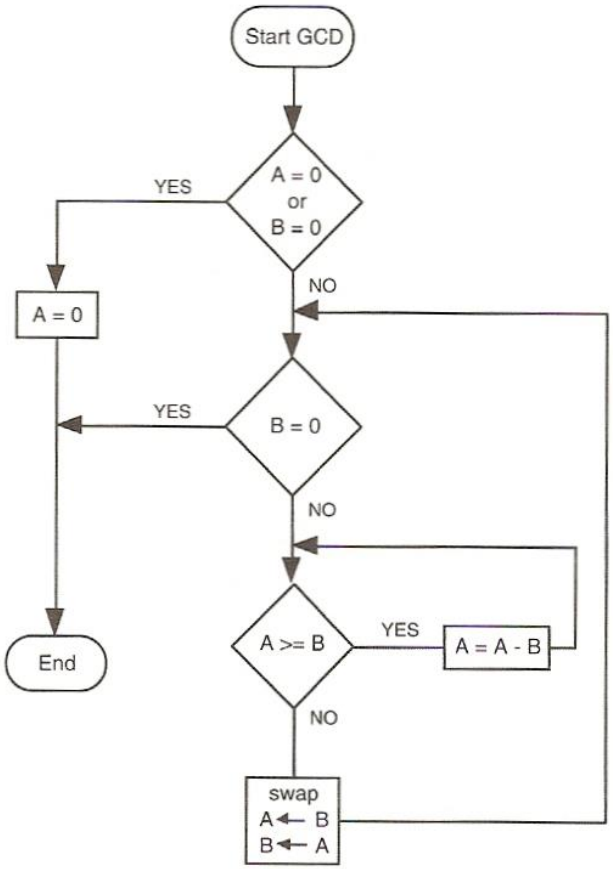


- Řídicí logika (kombinační logická síť)
  - Může být implementována pomocí paměti (např. ROM, viz dále)
  - Chování automatu může tedy být naprogramováno
- Datová cesta
  - Typicky z hradel - implementace pomocí paměti nemusí být efektivní (viz např. násobička)

Implementace algoritmu: Příklad

- Nejmenší společný dělitel (Greatest Common Divisor)
  - Popis v jazyku C

```
A = A_in;
B = B_in;
if (A != 0 && B != 0)
  {
    while (B != 0)
      {
        while (A >= B)
          {
            A = A - B;
          }
        Swap = A;
        A = B;
        B = Swap;
      }
  }
else
  {
    A = 0;
  }
Y = A;
```



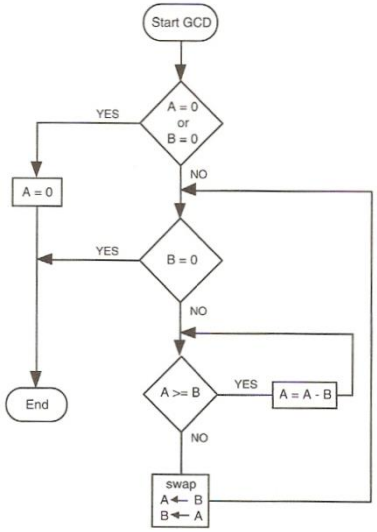
  - Behaviorální popis ve VHDL

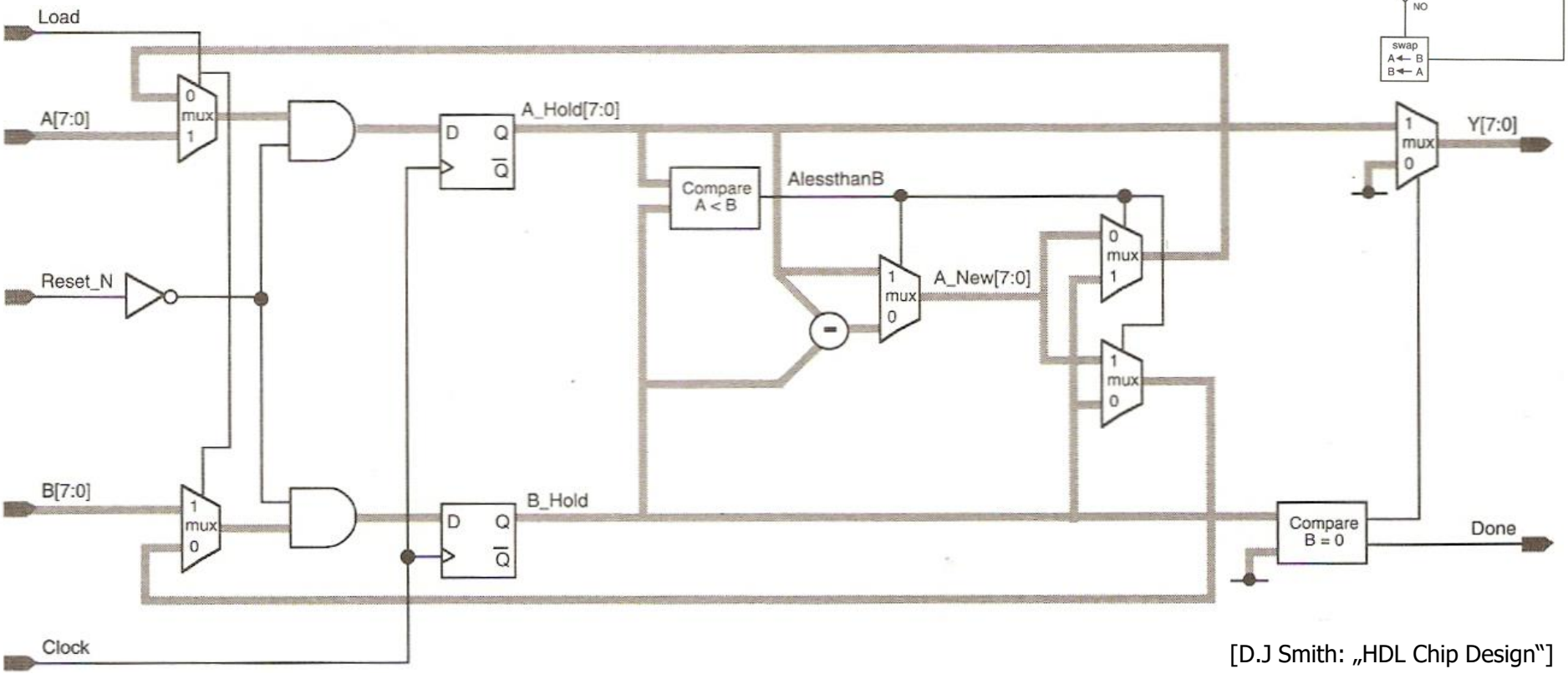
```
A := A_in;
B := B_in;
if (A /= 0 and B /= 0) then
   while (B /= 0) loop
      while (A >= B) loop
         A := A - B;
      end loop;
      Swap := A;
      A    := B;
      B    := Swap;
   end loop;
else
   A := 0;
end if;
Y := A;
```

[D.J Smith: „HDL Chip Design"]

Implementace algoritmu: Příklad

- Architektura GCD obvodu na RTL úrovni





[D.J Smith: „HDL Chip Design"]

Implementace algoritmu: Příklad

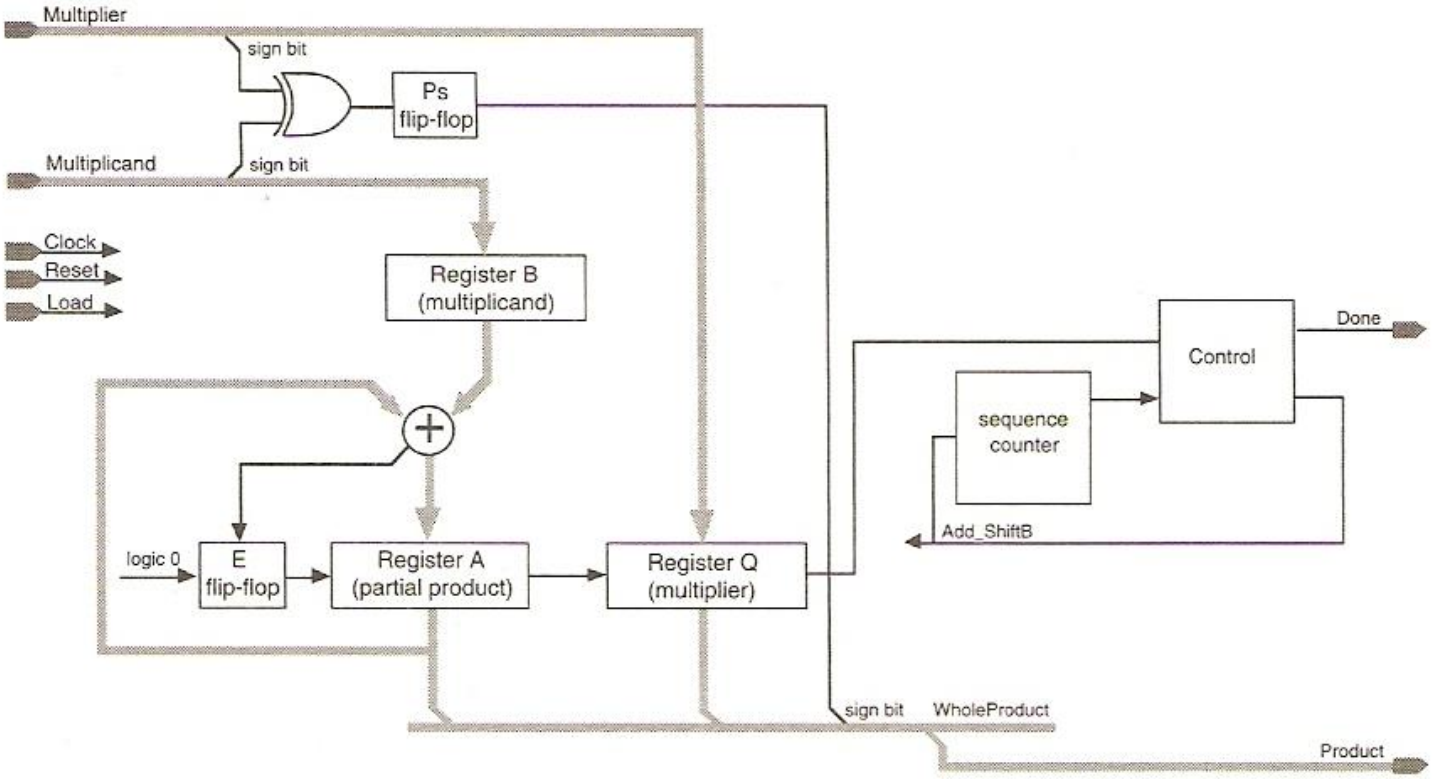
- Popis GCD obvodu ve VHLD na RTL úrovni

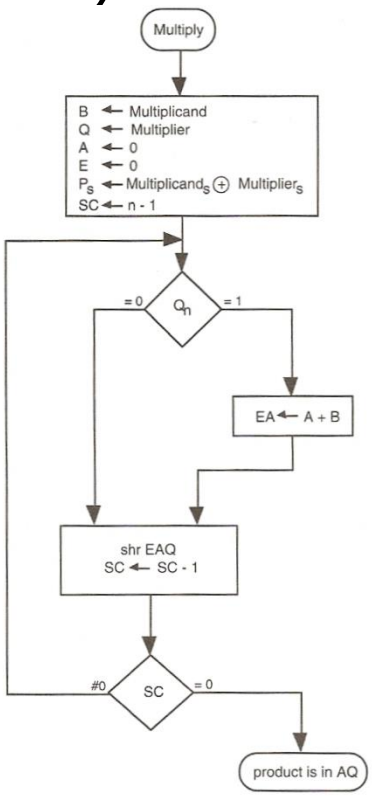
```
library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;

entity GCD is
    generic (Width: natural);
    port ( Clock, Reset, Load: in  std_logic;
           A, B:       in   unsigned(Width - 1 downto 0);
           Done:       out  std_logic;
           Y:          out  unsigned(Width - 1 downto 0));
end entity GCD;

architecture RTL of GCD is
    signal A_New,A_Hold,B_Hold: unsigned(Width-1 downto 0);
    signal A_lessthan_B: std_logic;
begin
-------------------------------------------------------------------
-- Load 2 input registers and ensure B_Hold < A_Hold
-------------------------------------------------------------------
LOAD_SWAP: process (Clock)
begin
   if rising_edge(Clock) then
      if (Reset = '1') then
         A_Hold <= (others => '0');
         B_Hold <= (others => '0');
      elsif (Load = '1') then
         A_Hold <= A;
         B_Hold <= B;
      elsif (A_lessthan_B = '1') then
         A_Hold <= B_Hold;
         B_Hold <= A_New;
      else
         A_Hold <= A_New;
      end if;
   end if;
end process LOAD_SWAP;
```

```
-------------------------------------------------------------------
-- Subtract B_Hold from A_Hold if A_Hold >= B_Hold
-------------------------------------------------------------------
SUBTRACT: process (A_Hold, B_Hold)
begin
   if (A_Hold >= B_Hold) then
      A_lessthan_B  <= '0';
      A_New <= A_Hold - B_Hold;
   else
      A_lessthan_B  <= '1';
      A_New <= A_Hold;
   end if;
end process SUBTRACT;

-------------------------------------------------------------------
-- Greatest common divisor found if B_Hold = 0
-------------------------------------------------------------------
WRITE_OUTPUT: process (A_Hold, B_Hold)
begin
   if (B_Hold = (others => '0')) then
      Done <= '1';
      Y <= A_Hold;
   else
      Done <= '0';
      Y <= (others => '0');
   end if;
end process WRITE_OUTPUT;

end architecture RTL;
```

Implementace algoritmu: Příklad

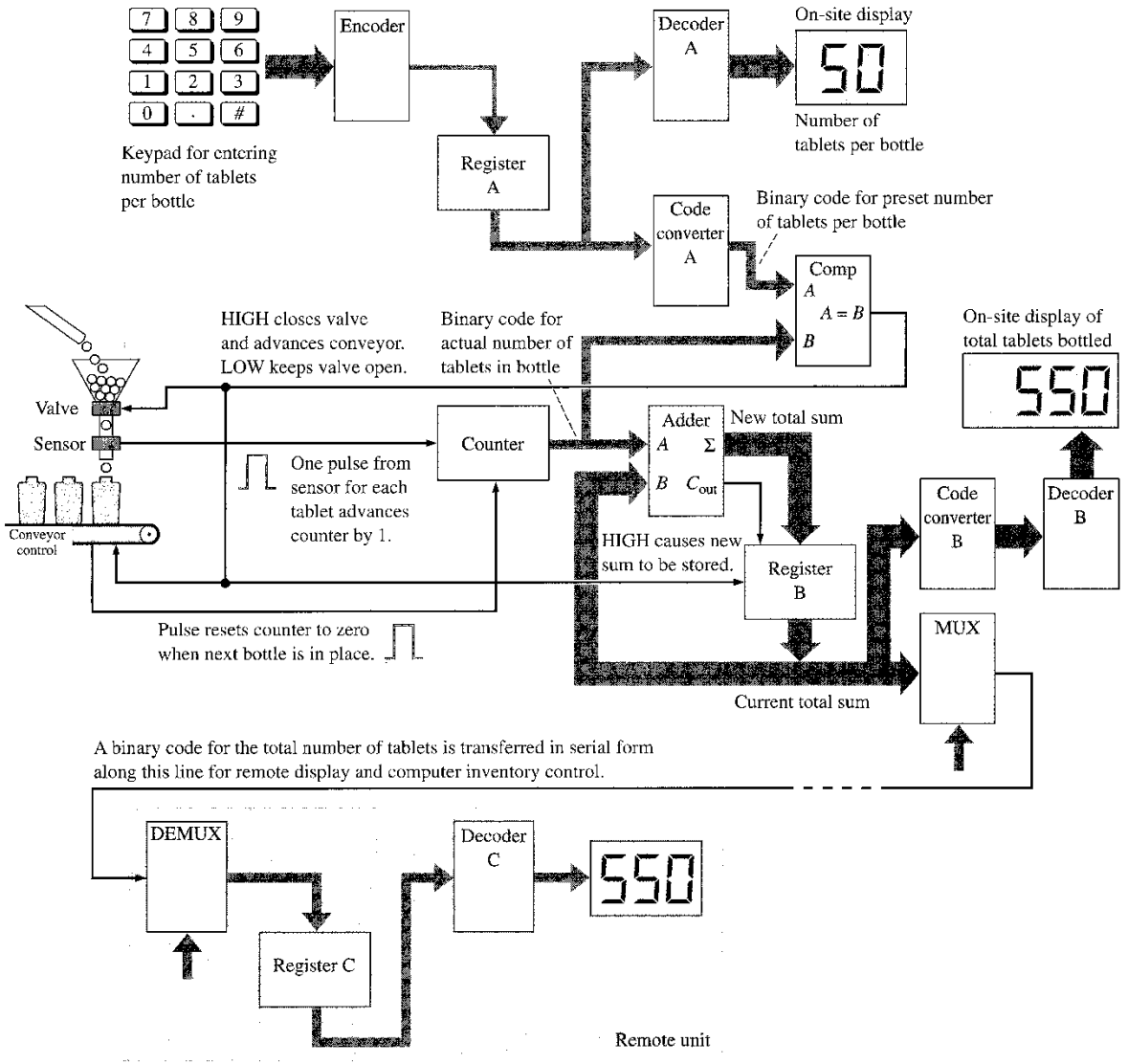
- Sekvenční násobička ve dvojkovém doplňku (shift & add)



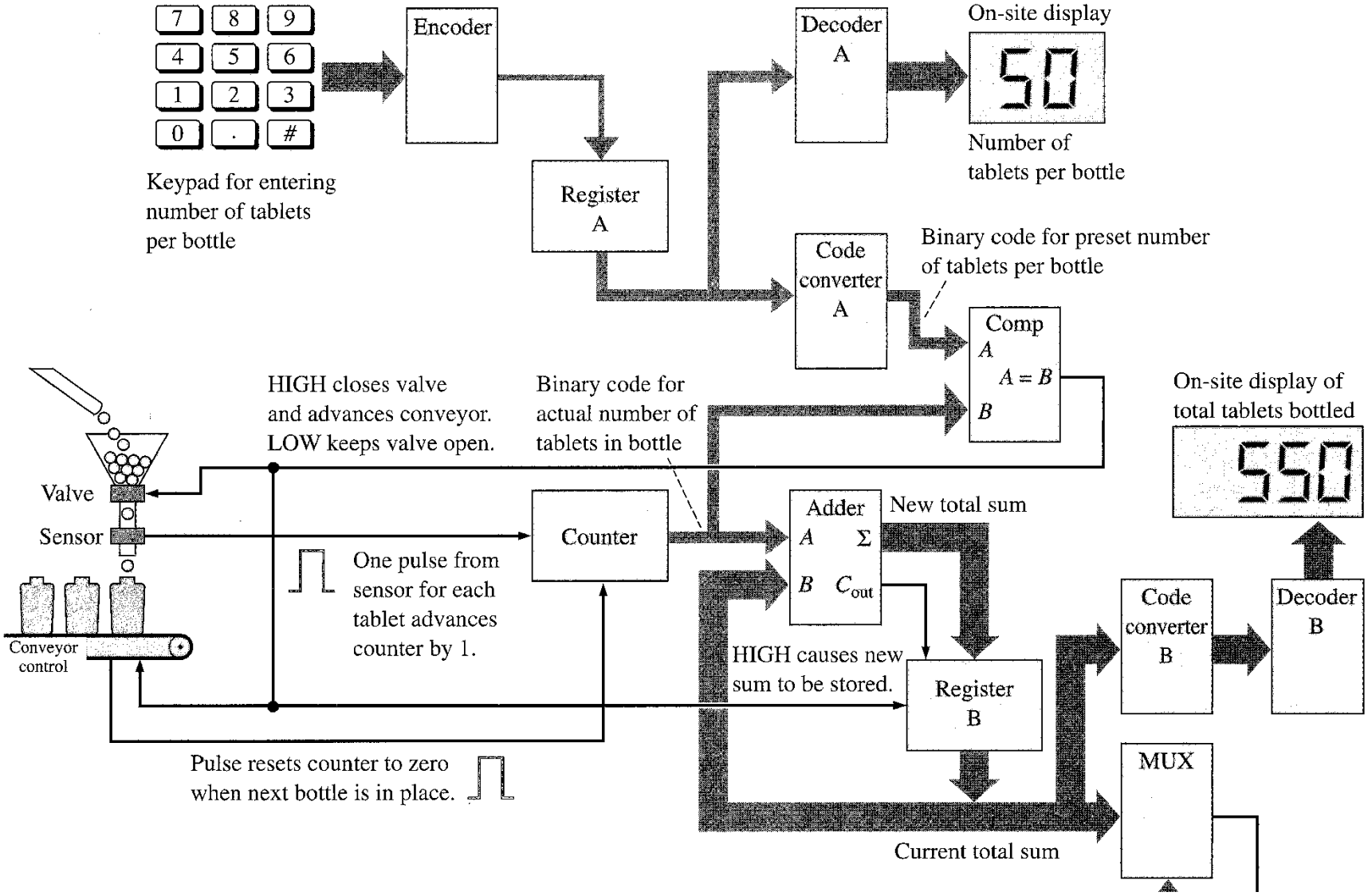


Implementace algoritmu: Aplikačně specifický obvod

- Plnička lékovek
  - pásový dopravník
  - plnicí jednotka
    - klapka
    - senzor
  - řídicí jednotka
    - klávesnice
    - kodér klávesnice
    - registry
    - dekodéry displejů
    - kodéry
    - komparátor
    - sčítačka
    - multiplexor
    - displej počtu pilulek na lékovku
    - displeje celkového počtu pilulek
  - kontrolní jednotka
    - na vzdáleném pracovišti
    - kontrola celkového počtu pilulek

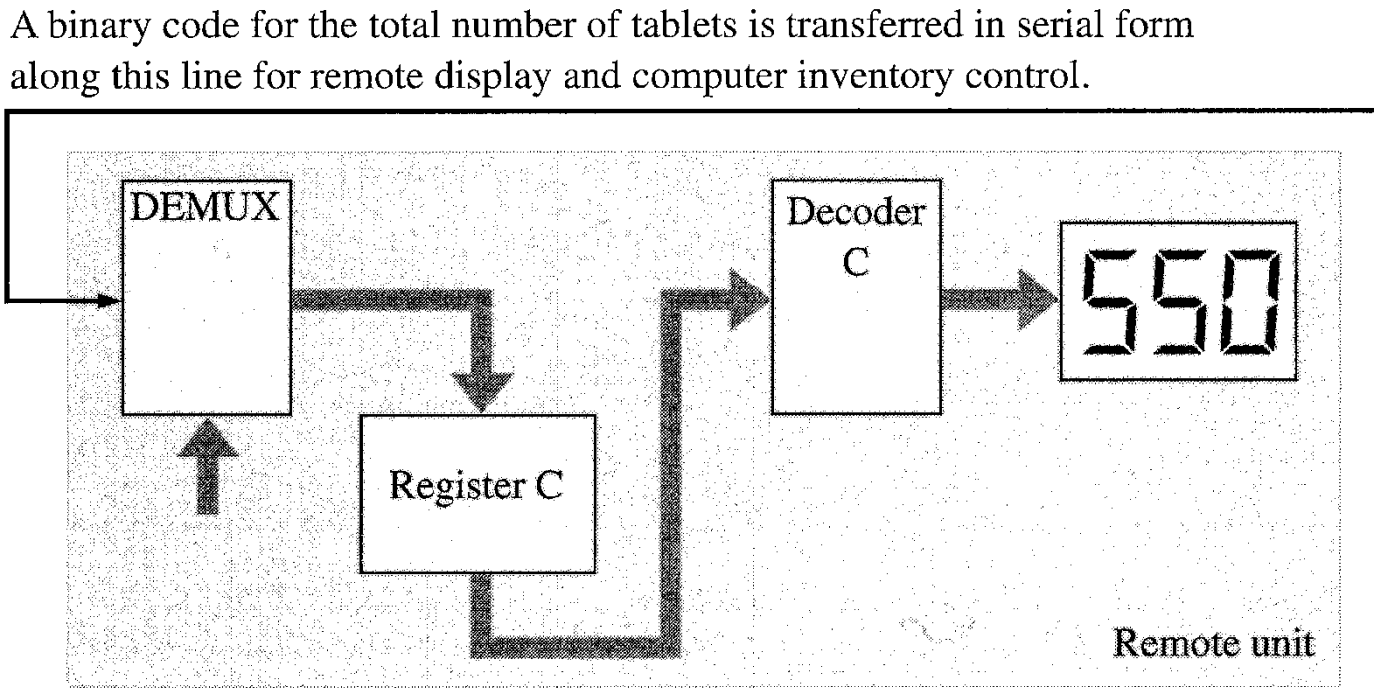


Plnička lékovek: Plnicí část

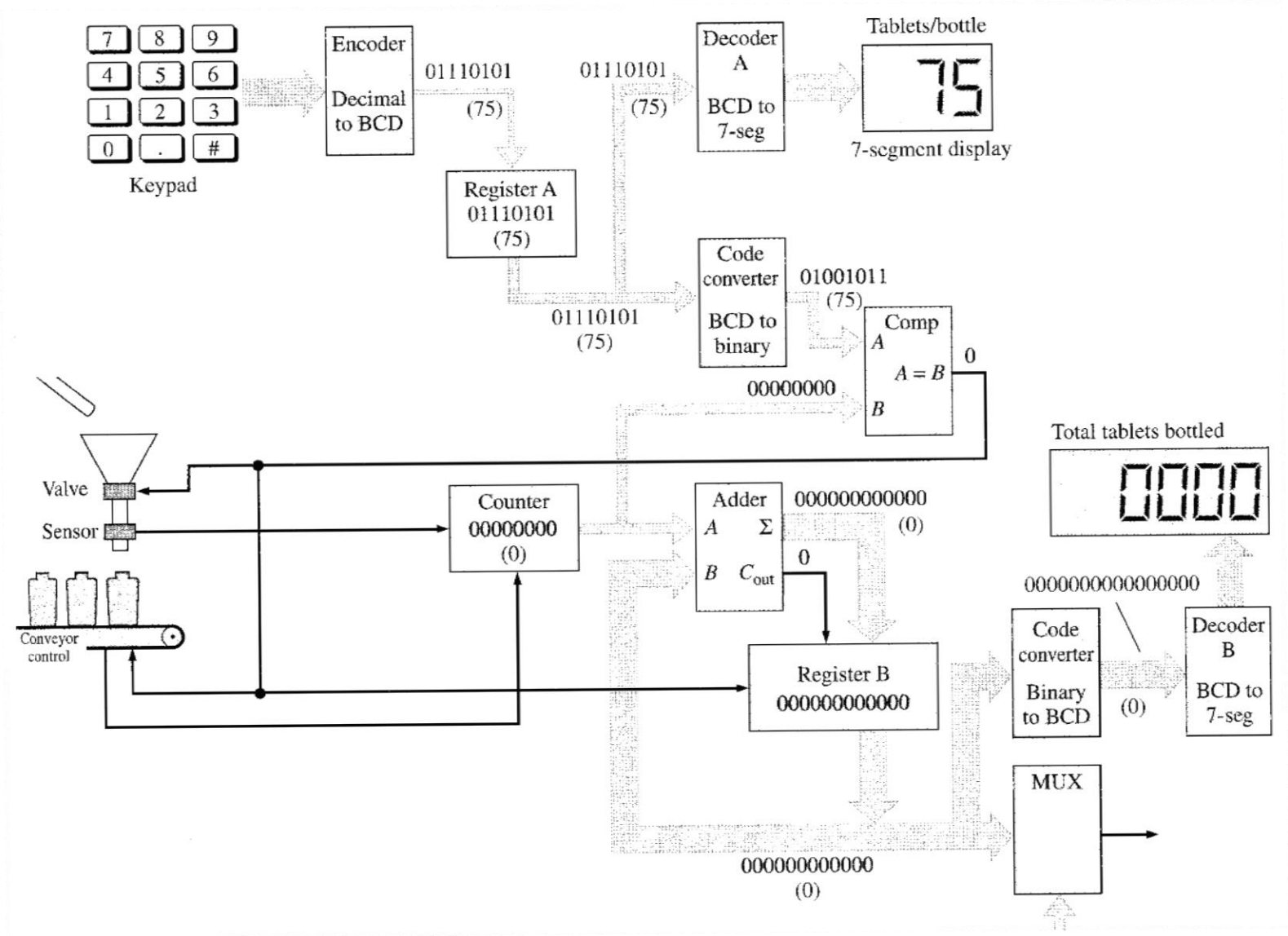


Plnička lékovek: Vzdálené kontrolní pracoviště

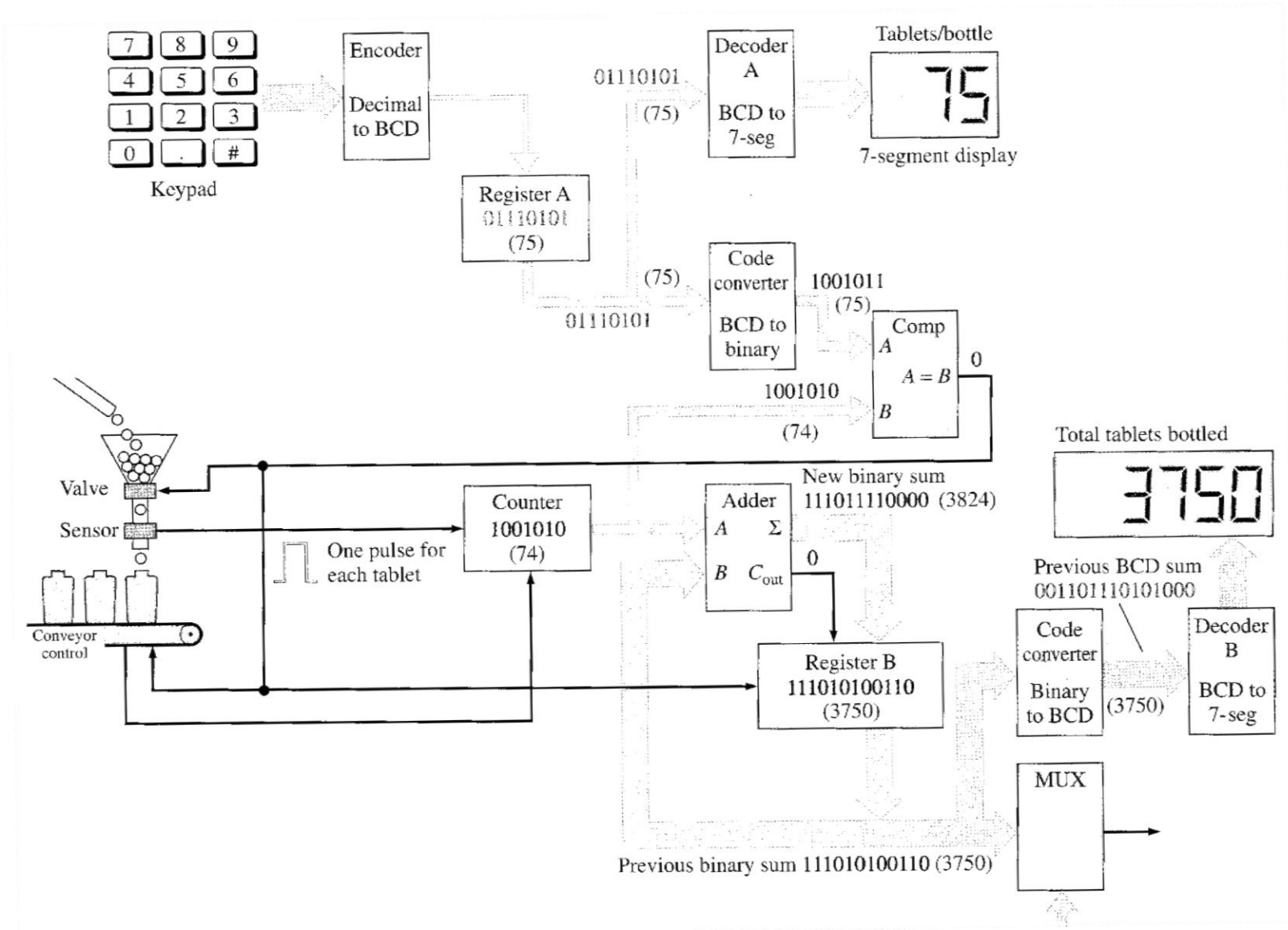
- Slouží pro kontrolu celkového počtu naplněných pilulek
  - je umístěna na vzdáleném pracovišti
  - přenos údajů v sériové formě
  - pomocí MUX a DMUX



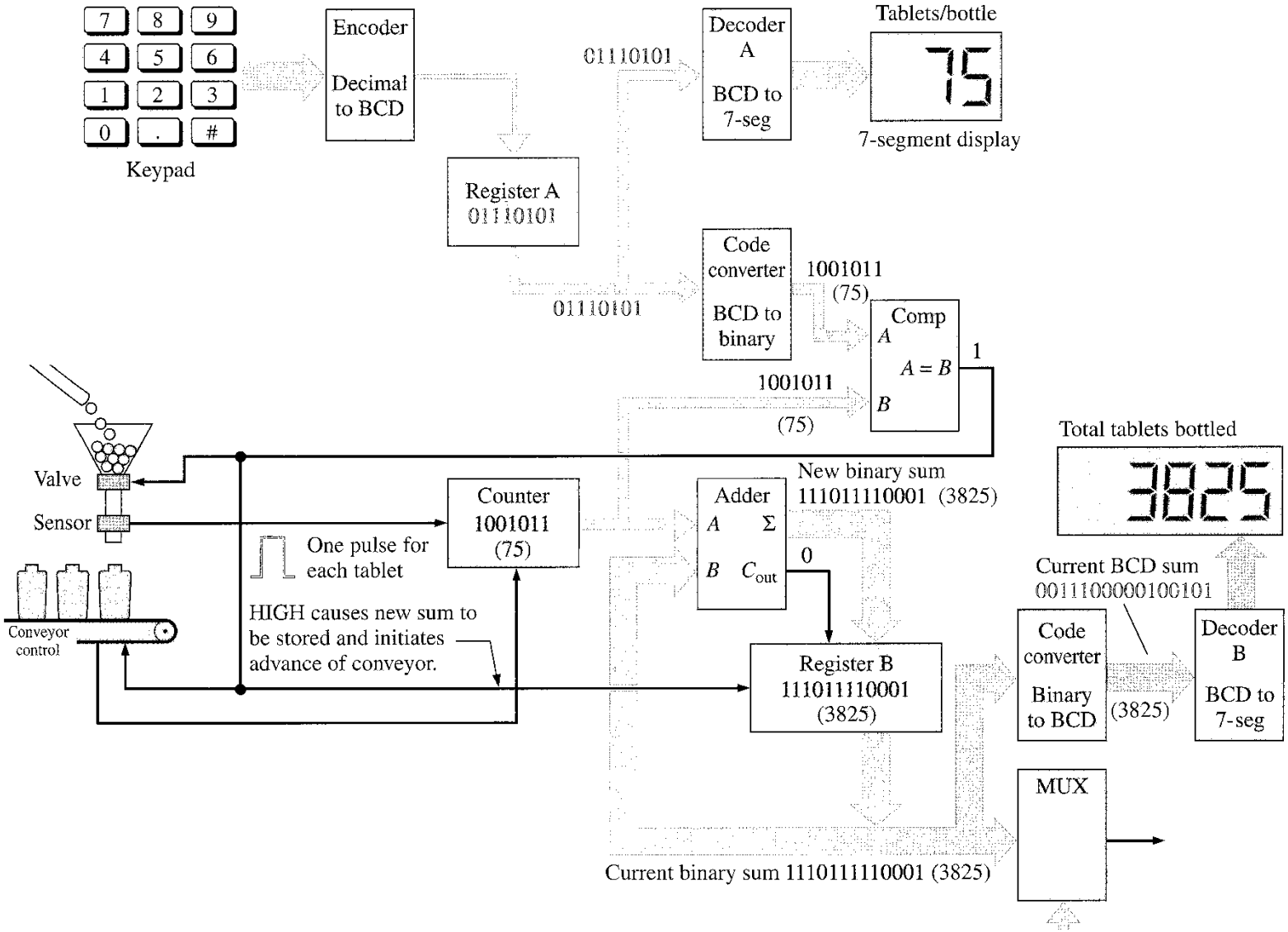
Plnička lékovek: Stav po resetu



Plnička lékovek: Je právě plněna lékovka č.51

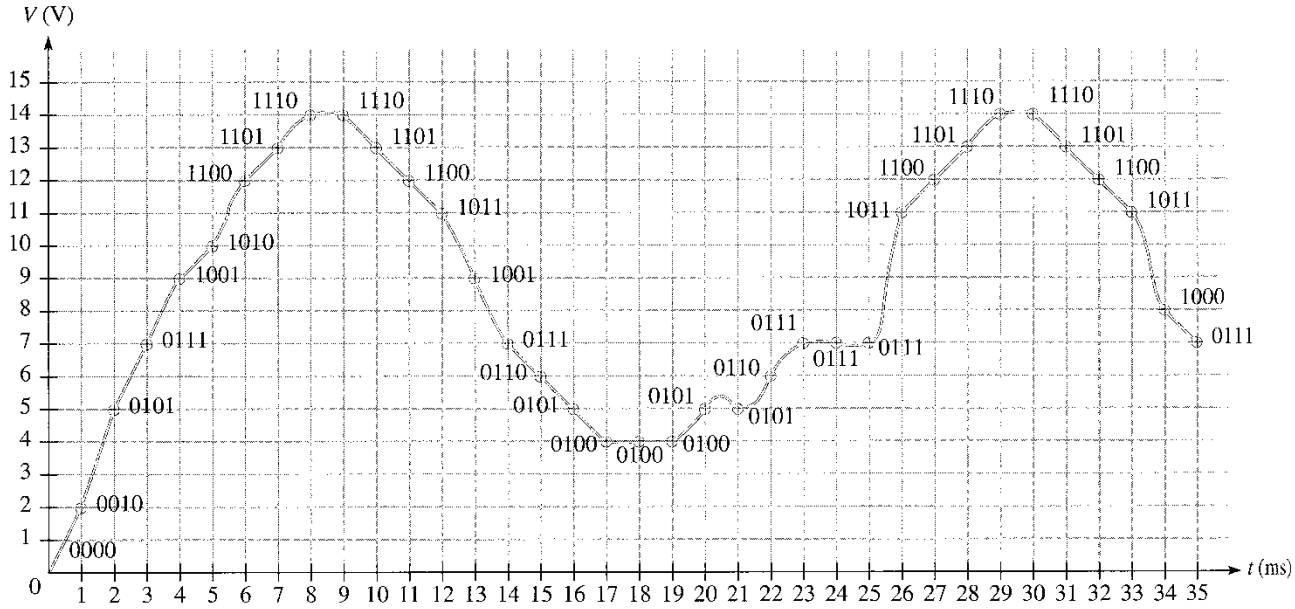


Plnička lékovek: Je právě naplněna lékovka č.51



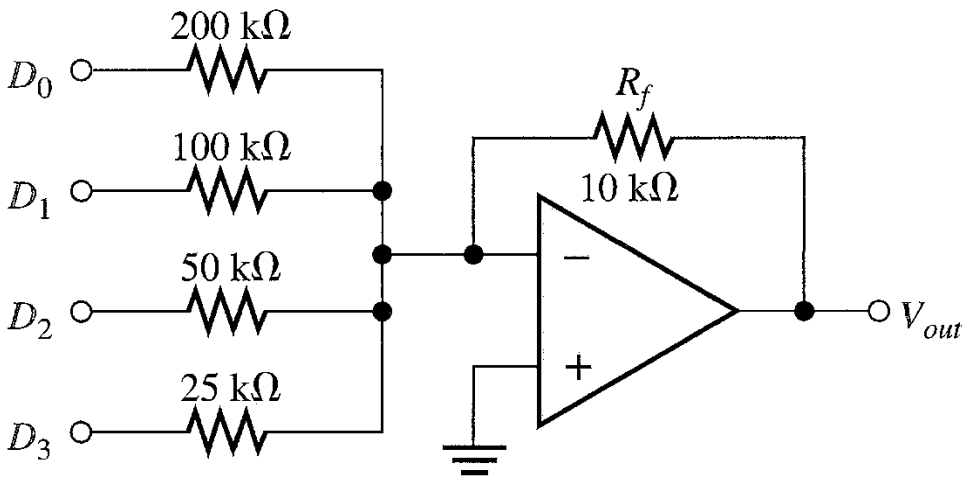
Analogové vs. číslicová informace

- Analogové veličiny je třeba převést před jejich digitálním zpracováním do číslicové podoby a po jejich zpracování zpět na analogové
  - Počet bitů převedeného čísla udává přesnost
  - Počet vzorků udává frekvenci
  - Digitálně/Analogový (D/A) – „Digital-to-Analog Converter"
  - Analogově/Digitální (A/D) – „Analog-to-Digital Converter"



Digitálně - analogový převodník

- Příklad realizace
  - Na vstupech D0..D3 je buď 0V, nebo 5V (log.0 nebo log.1)
  - Výstupní napětí je dáno součtem napětí jednotlivých Vout(Di) pro Di v log.1 (pro i=0..3)
  - Převádí 4-bitovou binární informaci a 16 různých napěťových úrovní – analogová informace

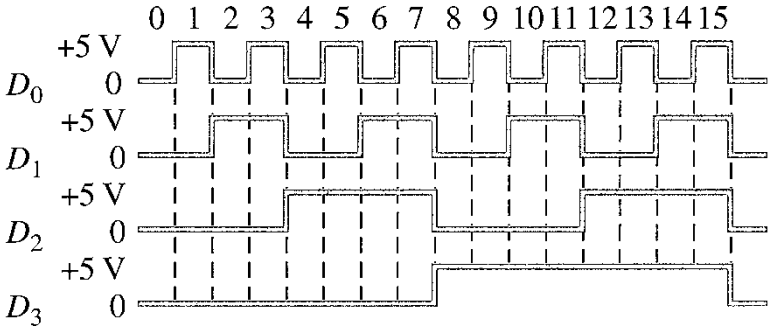


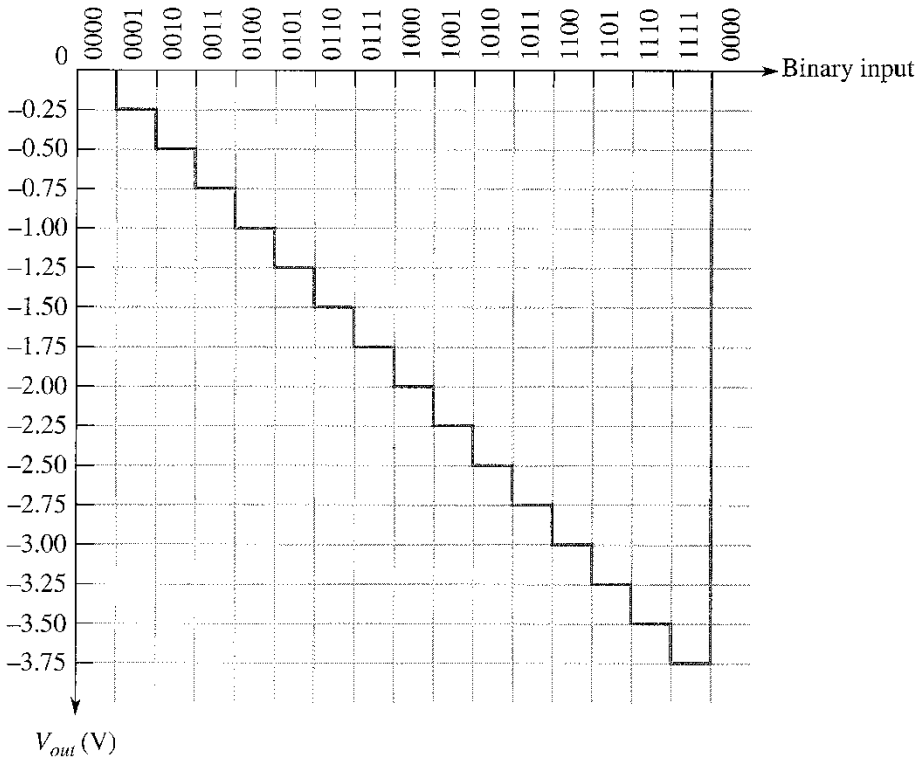
$$V_{out(D0)} = (10\ \text{k}\Omega)(-0.025\ \text{mA}) = -0.25\ \text{V}$$
$$V_{out(D1)} = (10\ \text{k}\Omega)(-0.05\ \text{mA}) = -0.5\ \text{V}$$
$$V_{out(D2)} = (10\ \text{k}\Omega)(-0.1\ \text{mA}) = -1\ \text{V}$$
$$V_{out(D3)} = (10\ \text{k}\Omega)(-0.2\ \text{mA}) = -2\ \text{V}$$

$$I_0 = \frac{5\ \text{V}}{200\ \text{k}\Omega} = 0.025\ \text{mA}$$
$$I_1 = \frac{5\ \text{V}}{100\ \text{k}\Omega} = 0.05\ \text{mA}$$
$$I_2 = \frac{5\ \text{V}}{50\ \text{k}\Omega} = 0.1\ \text{mA}$$
$$I_3 = \frac{5\ \text{V}}{25\ \text{k}\Omega} = 0.2\ \text{mA}$$

Digitálně - analogový převodník

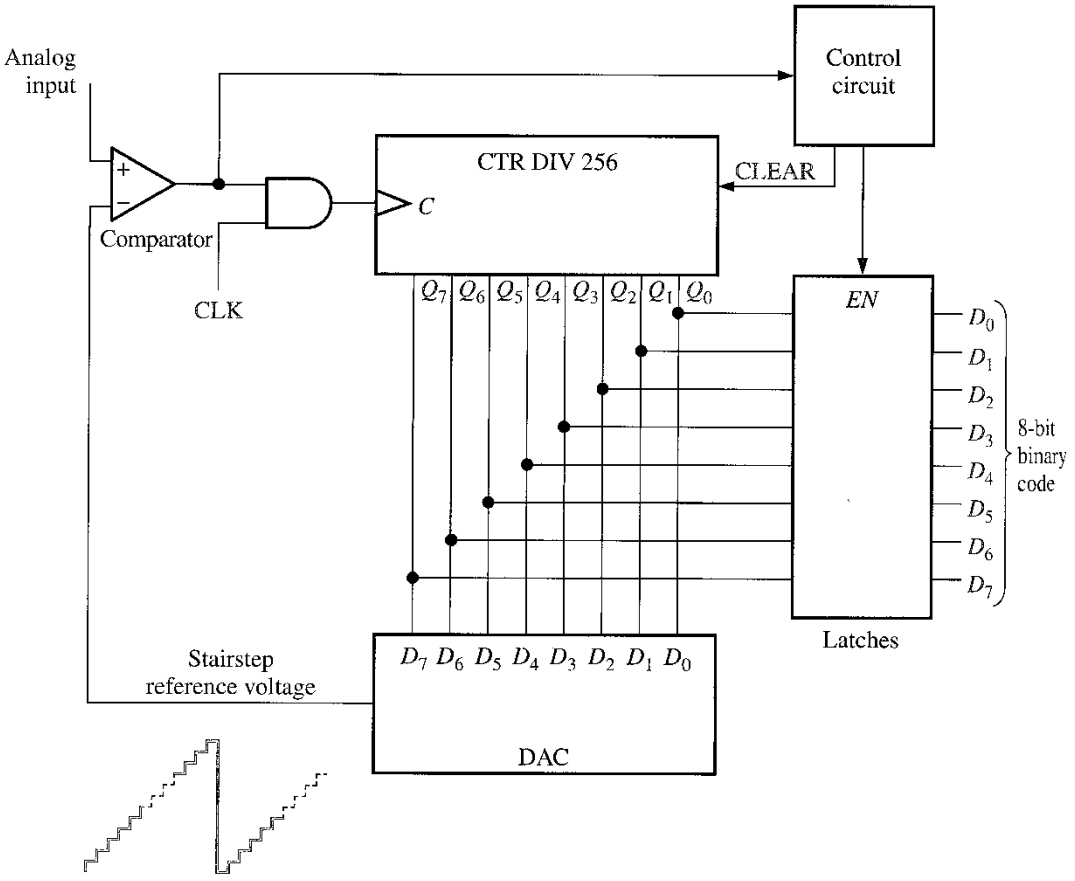
- Časový diagram
  - Vstupní číslicová informace
- Hodnoty na výstupu převodníku
  - Každé binární hodnotě na vstupu odpovídá hodnota analogová
  - Rozsah hodnot je (0V) [0000] až (-3,75V) [1111]





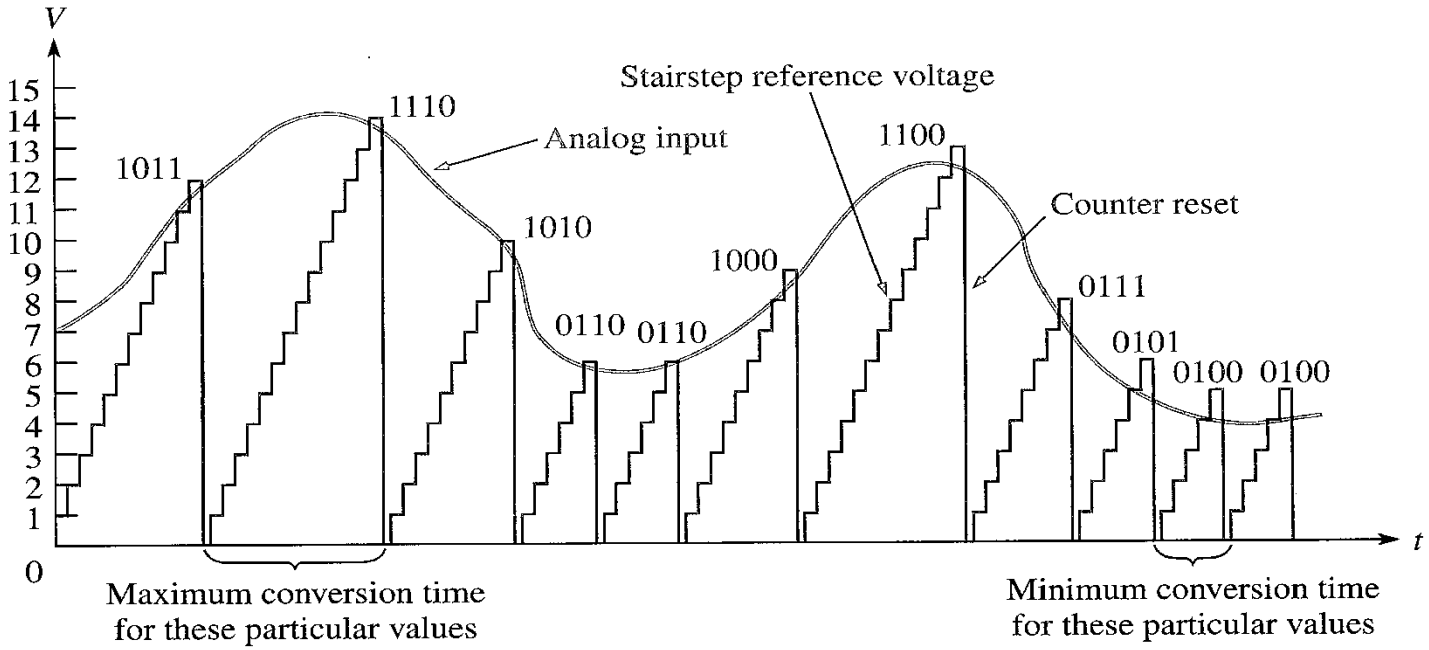
Analogově - digitální převodník: Generující pilu

- Čítač modulo 256
  - 8-mi bitový převod
- Komparátor
  - Porovnává generované napětí se vstupním napětím
- Řídicí obvod
  - Uloží aktuální stav čítače do výstupního registru, jakmile je vstupní hodnota napětí rovna aktuální hodnotě čítače
- DAC – D/A převodník
  - Převádí stav čítače na analogové napětí



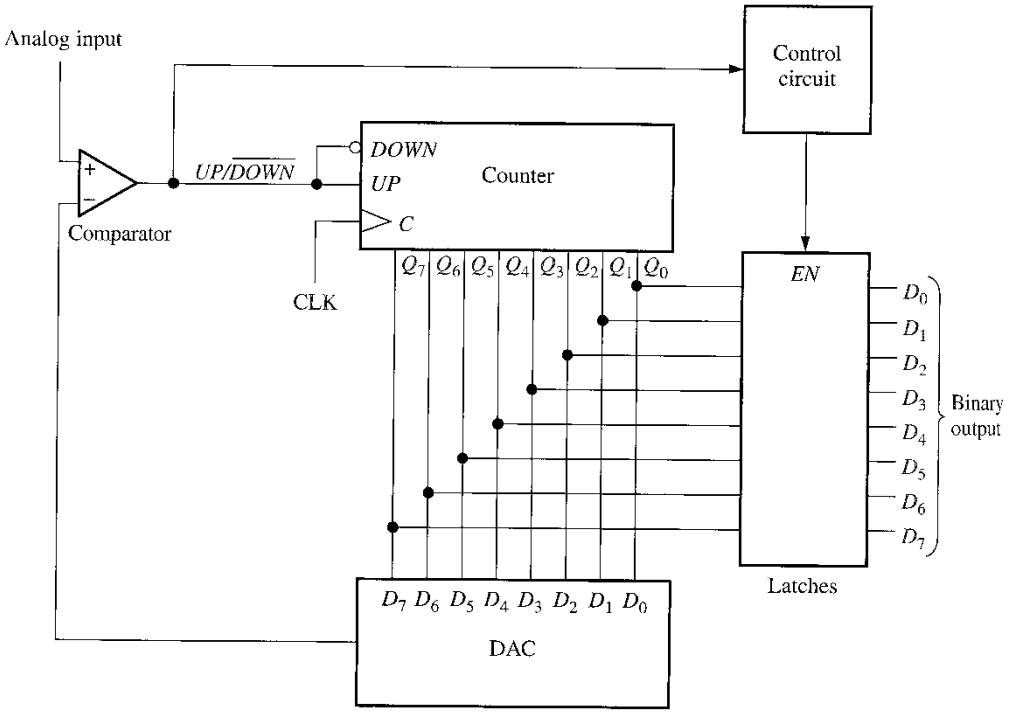
Analogově - digitální převodník: Generující pilu

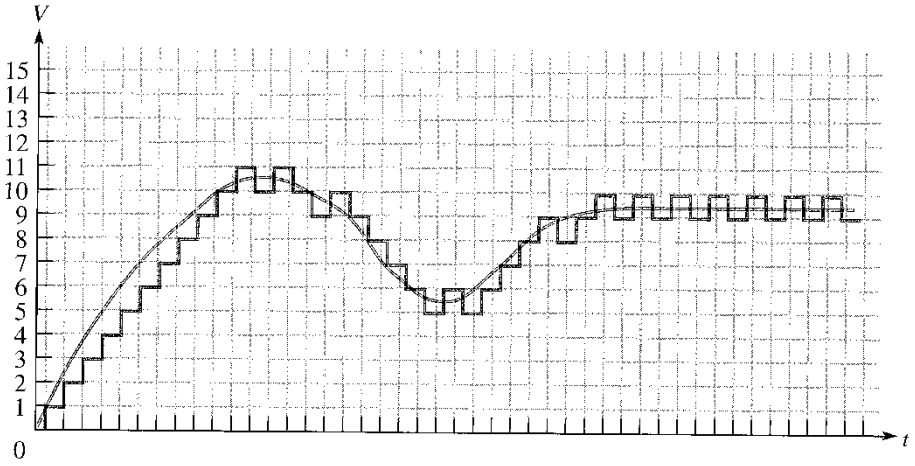
- Princip činnosti
  - D/A převodník generuje pilový průběh = postupně zvyšující se analogové napětí
  - Jakmile je generované pilové napětí ≥ vstupní napětí, zastaví se generátor pily a jeho hodnota odpovídá vstupnímu napětí
  - Doba převodu je úměrná velikosti vstupního napětí
  - Po každém převodu se čítač nuluje



Analogově - digitální převodník: Sledující

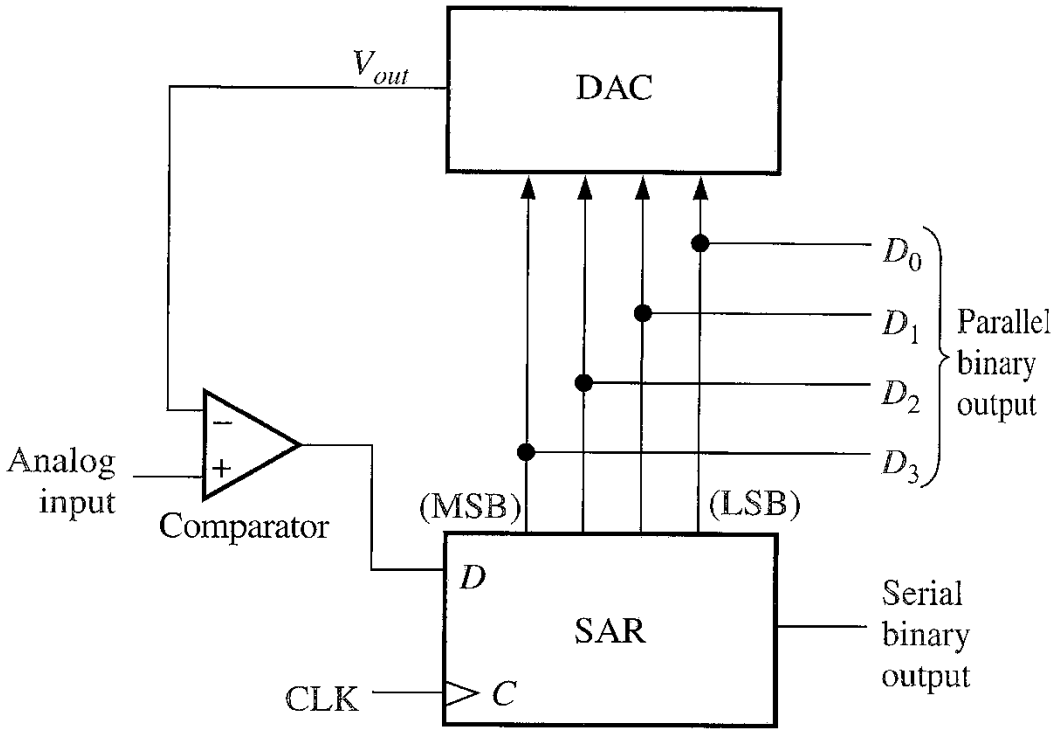
- Princip činnosti
  - DAC generuje pilový průběh = postupně zvyšující se analogové napětí
  - Jakmile je generované pilové napětí ≥ vstupní napětí, zastaví se generátor pily a jeho hodnota odpovídá vstupnímu napětí
  - Jakmile vstupní napětí opět vzroste, čítač bude čítat nahoru až do hodnoty odpovídající vstupnímu napětí
  - Pokud vstupní napětí klesne, bude čítač čítat dolů až do hodnoty odpovídající vstupnímu napětí
  - Převodník sleduje vstupní napětí



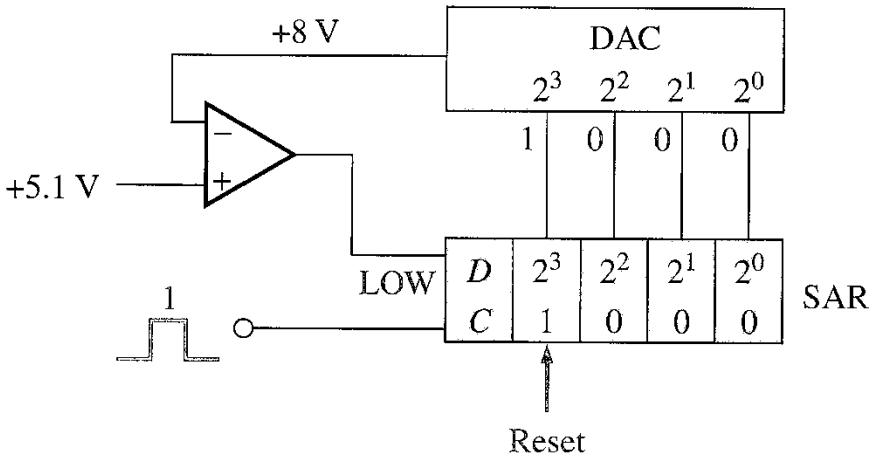


Analogově - digitální převodník: Aproximační

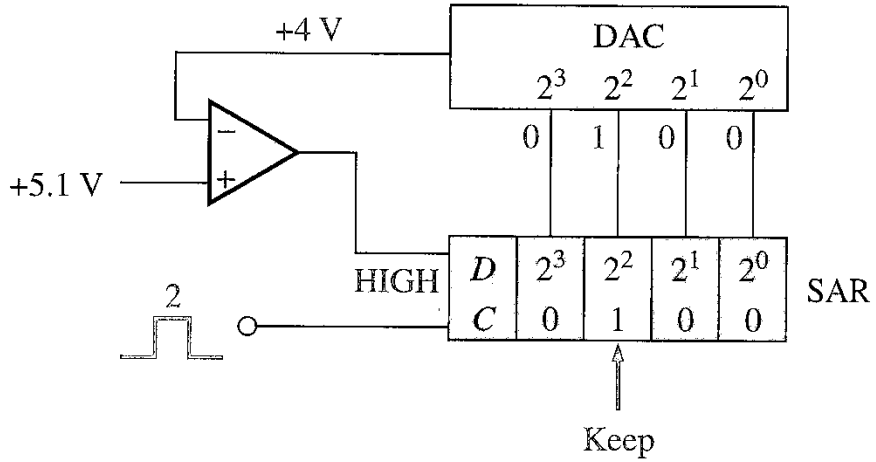
- Pracuje na principu dělení intervalu
  - Nejprve se nastaví MSB na log.1 => porovná se, zda je hodnota vstupního napětí menší či větší než polovina
  - Pokud je větší, v SAR se zapamatuje MSB=log.1
  - Pokud je menší, v SAR se zapamatuje MSB=log.0
  - Dále se pokračuje pro hodnoty v horní či dolní polovině, atd.
  - SAR – Successive-Approximation Register
  - Celkový počet kroků je roven počtu bitů
- Rychlé



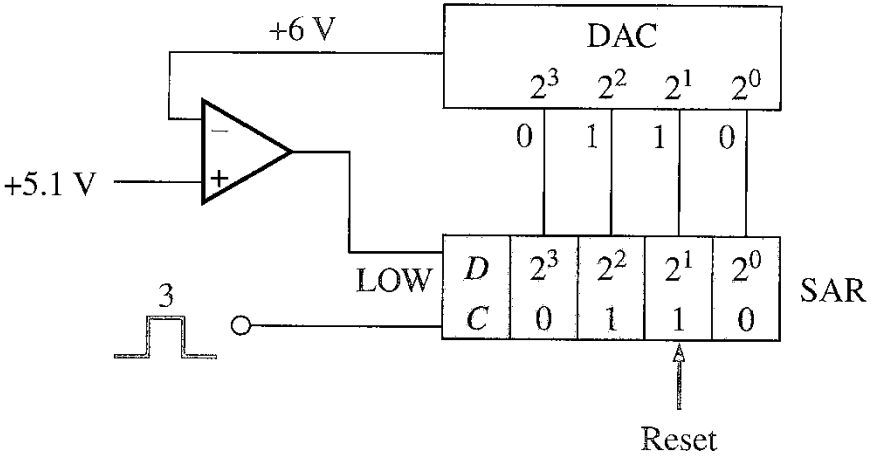
Analogově - digitální převodník: Aproximační



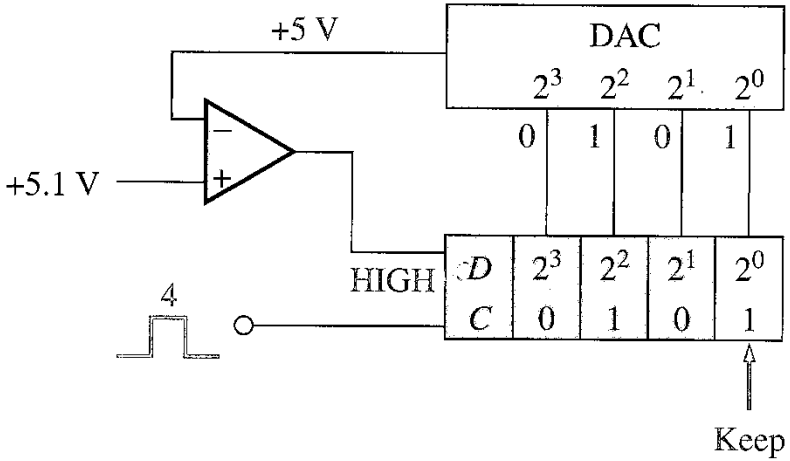
(a) MSB trial



(b) 2^2-bit trial



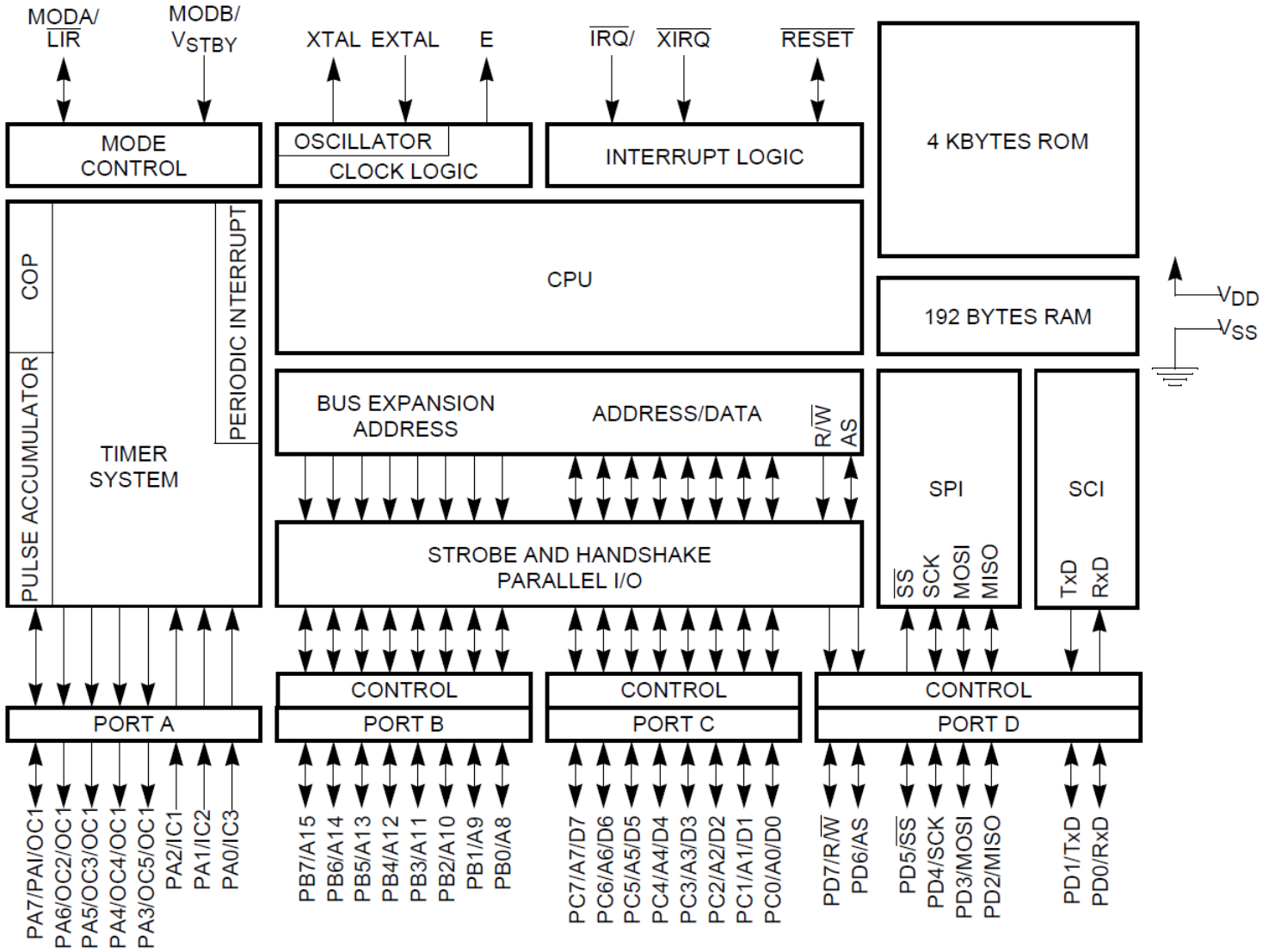
(c) 2^1-bit trial



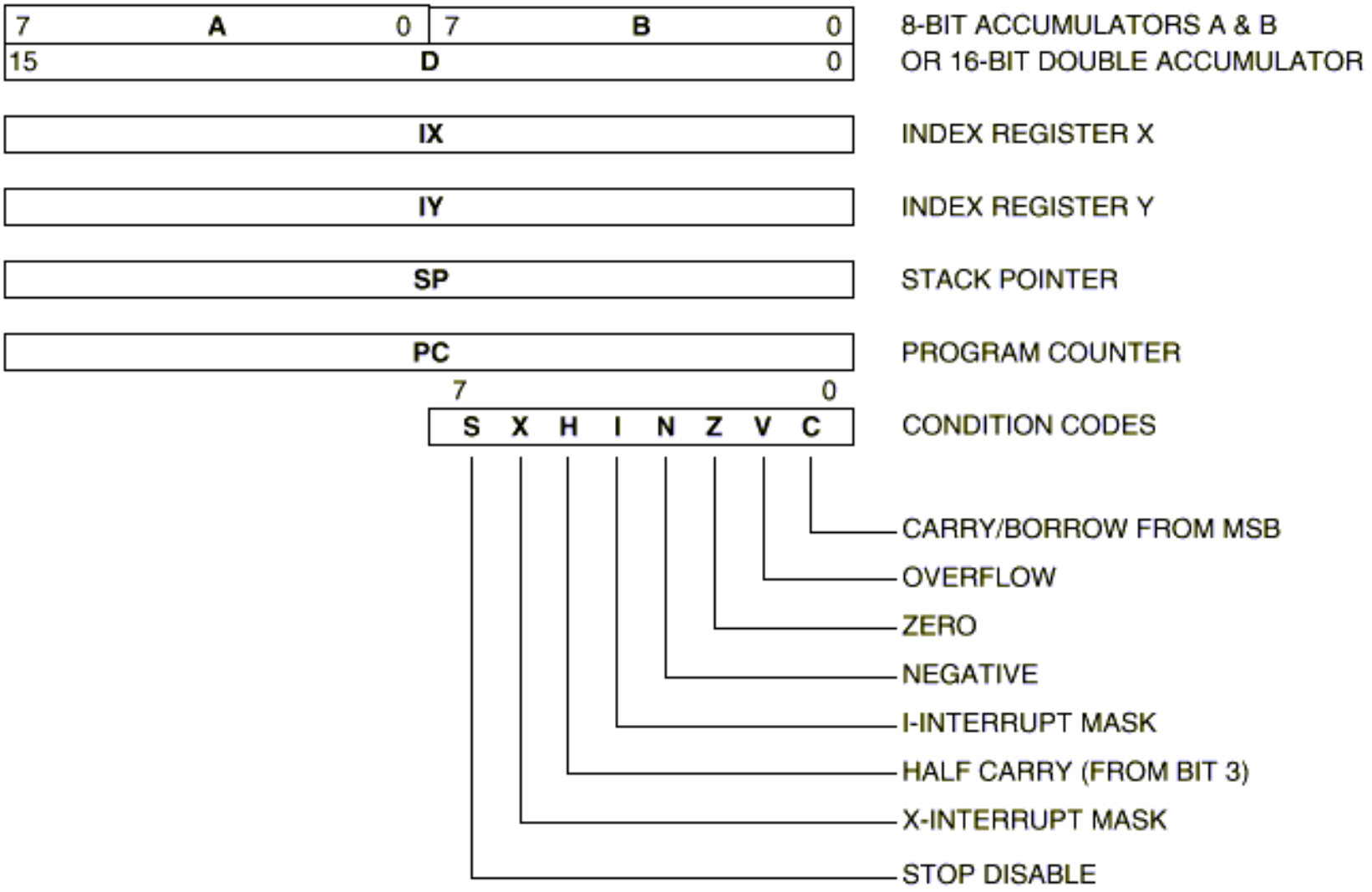
(d) LSB trial (conversion complete)

Příklad procesoru: Mikrokontrolér Motorola HC11

- Blokové schéma



HC11: Programátorský model

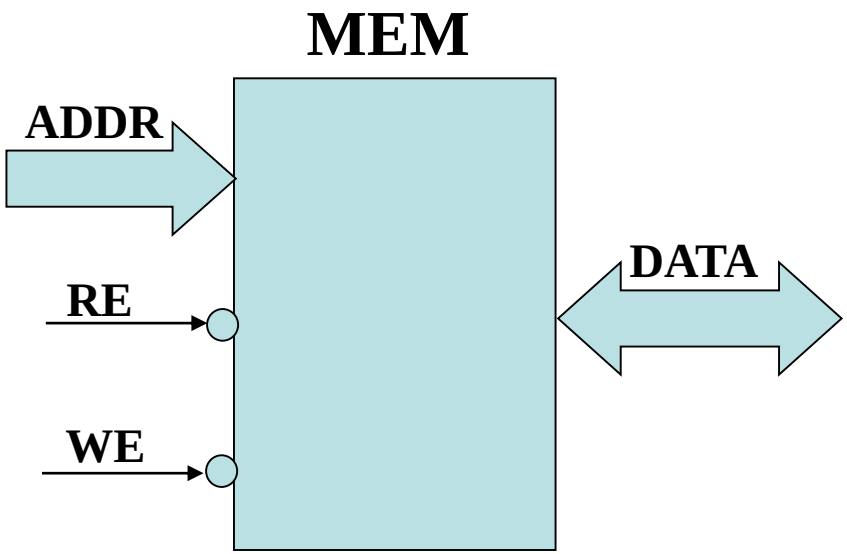


HC11: Základní bloky

- Paměť programu a dat
  - Program/Data Memory
- Aritmetická a logická jednotka
  - Arithmetic Logic Unit (ALU)
- Sada registrů
  - Register Bank (ACCA, ACCB, IX, IY, SP)
- Programový čítač
  - Program Counter (PC)
- Sčítačka
  - Adder (ADD)
- Registr instrukcí
  - Instruction Register (IR)
- Řadič procesoru
  - Controller (CNTL)
- Registr příznaků
  - Flag Register (CCR)
- Pomocné registry
  - Temporary Registers (ALU Result, Mem OpCode/Data)
- Datová cesta je realizována pomocí multiplexorů

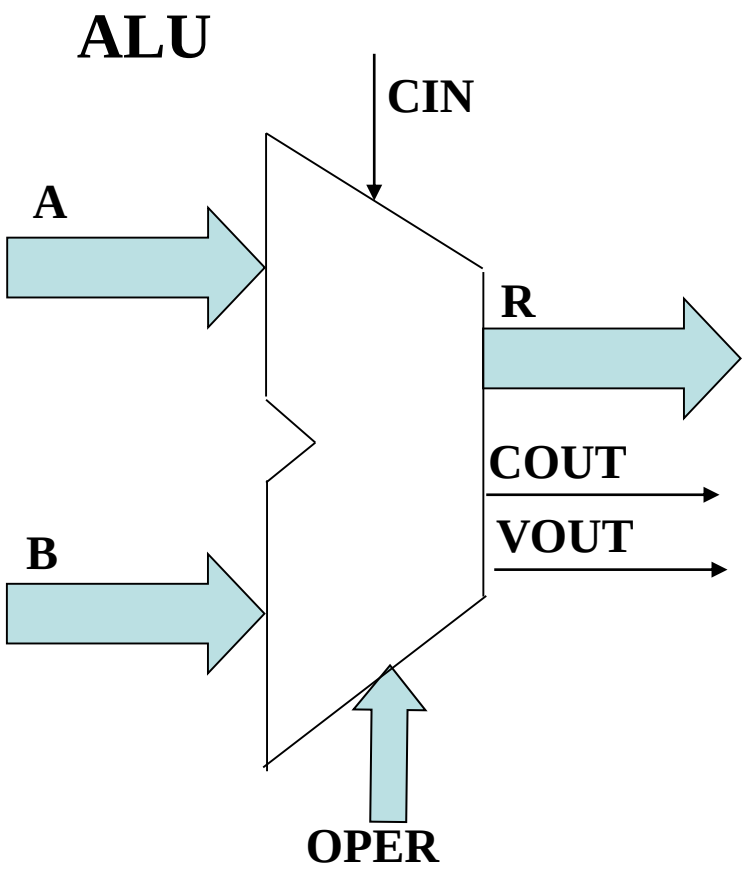
HC11: Paměť (Program/Data)

- Asynchronní SRAM
  - Address Bus (ADDR)
- 16-bitová adresa (64kB)
  - Data Bus (DATA)
- 8-bitová data (obousměrná sběrnice)
  - Write Enable (WE)
  - když WE = 0, tak zapiš DATA do paměti
  - zápis se provede na vzestupnou hranu WE
- Read Enable (RE)
  - když RE = 0, tak čti DATA z paměti

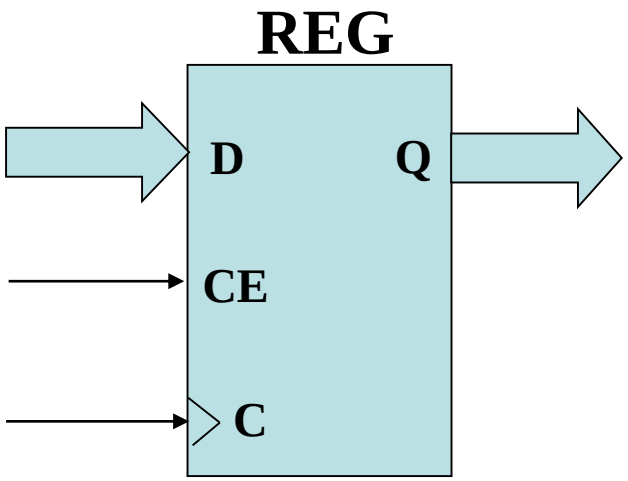


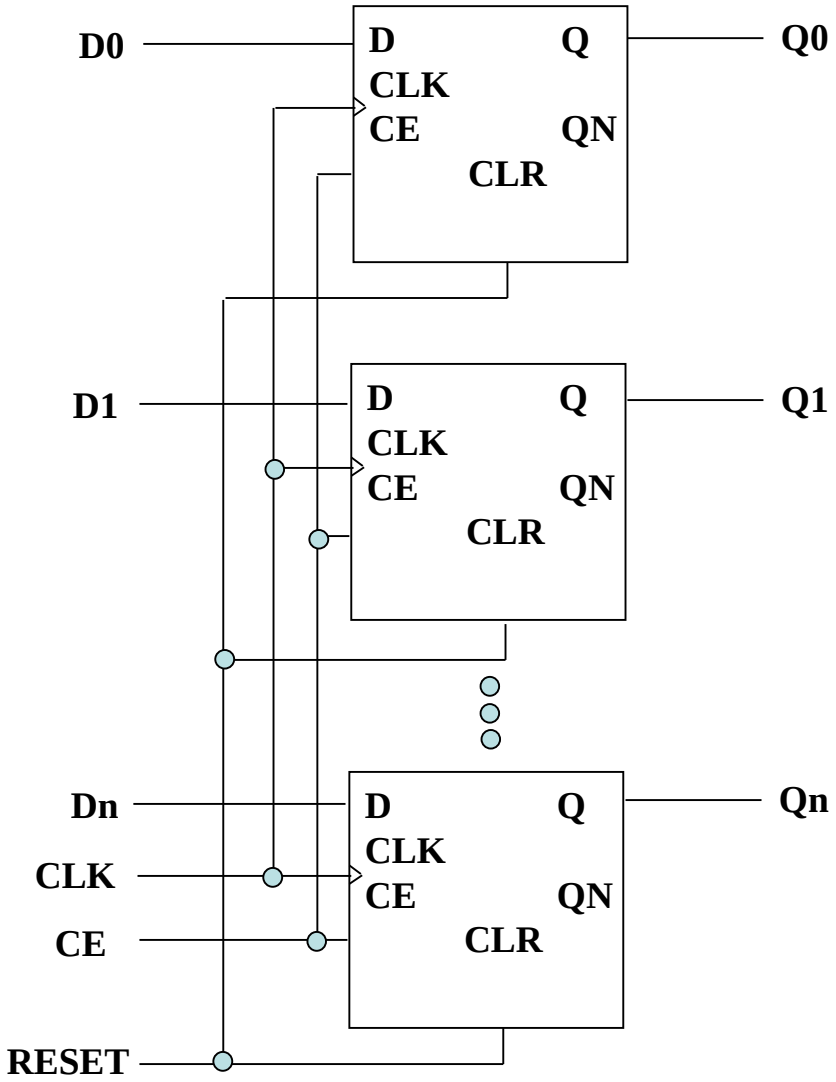
HC11: Aritmetická a logická jednotka (ALU)

- Signály
  - První vstupní operand (A)
  - Druhý vstupní operand (B)
  - Výstup součtu – Result (R)
  - Přenos z nižšího řádu - Carry In (CIN)
  - Přenos do vyššího řádu – Carry Out (COUT)
  - Detekce přetečení – Overflow (VOUT)
  - Výběr funkce – Operation (OPER)
    - Addition
    - Subtraction
    - Logic AND, OR, XOR
    - Shift Left/Right
    - Pass A/B – přenos data z A či B na R beze změny



- Jedno až 16bitové registry
  - Klopné obvody D se společnými řídícími vstupy
    - Synchronizační hodinový signál (CLK)
    - Asynchronní vstupy pro nastavení či nulování (PRE, CLR)
    - Povolení činnosti hodin (CE)
  - Vstupy (D0..Dn)
  - Výstupy (Q0..Qn)



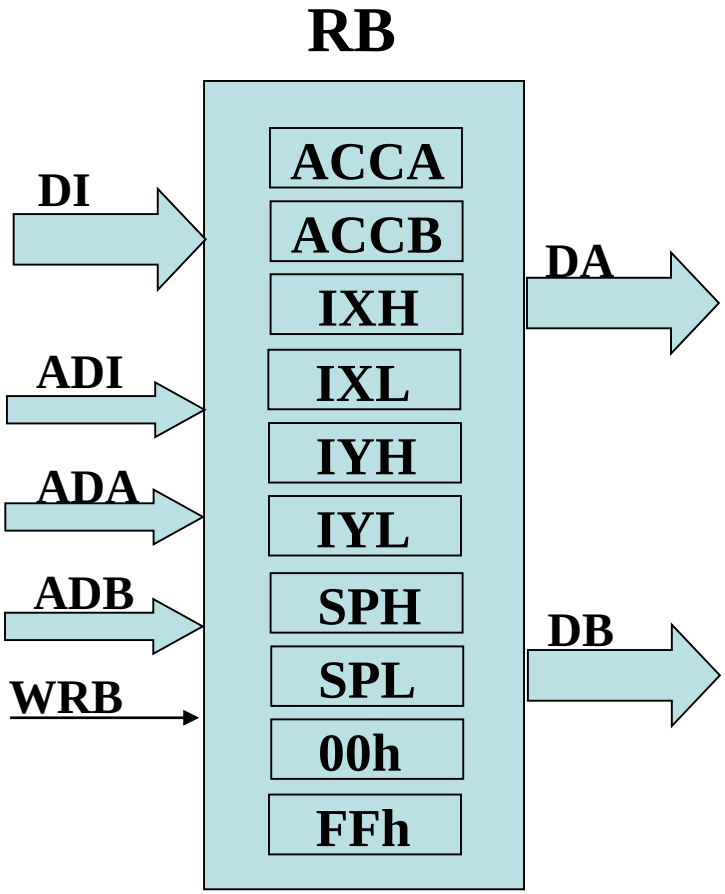


HC11: Registry

- Program Counter (PC)
  - Ukazatel do paměti programu (dat)
- Instruction Register (IR)
  - Operační kód (opcode) právě vykonávané instrukce
- Address Low (AL)
  - Nižší byte počítané adresy (např. pro Index Addressing Mode)
- Address High (AH)
  - Vyšší byte počítané adresy (např. pro Index Addressing Mode)
- Res (R)
  - Výsledek ALU
- Mem (M)
  - Data z paměti
- Temporary Carry (TC)
  - Využívá se pro 16-bitové ALU operace
- Carry Flag (C)
- Zero Flag (Z)
- Overflow Flag (V)

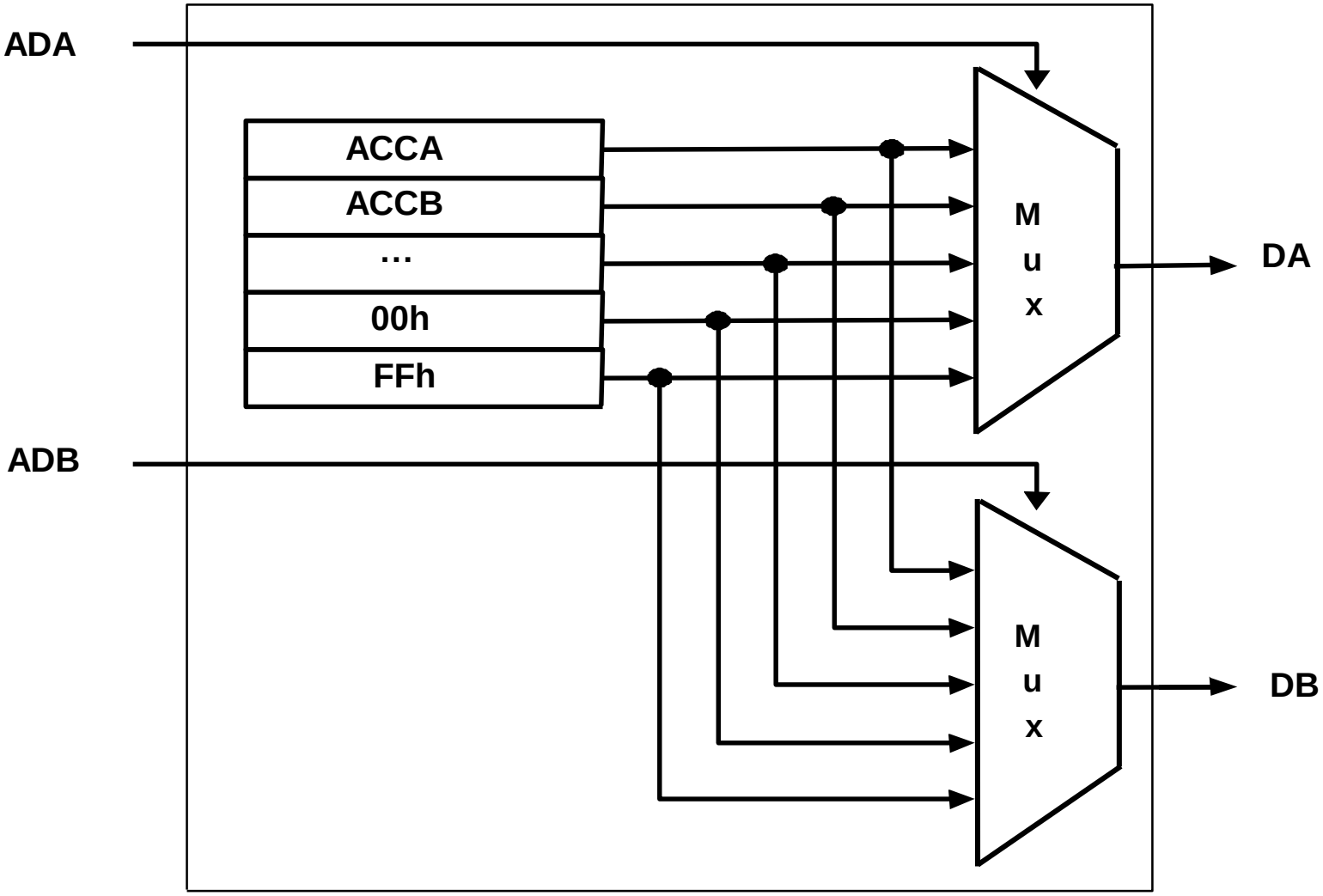
HC11: Sada registrů (Register Bank - RB)

- **Input Data (DI)**
  - vstupní datová sběrnice - input data bus
- **Input Data Address (ADI)**
  - adresa registru, do kterého se mají uložit data z DI
- **Output Data Bus A (DA)**
  - výstupní sběrnice A
- **Output Data A Address (ADA)**
  - adresa registru, ze kterého se čte na výstup DA
- **Output Data Bus B (DB)**
  - výstupní sběrnice B
- **Output Data B Address (ADB)**
  - adresa registru, ze kterého se čte na výstup DA
- **Write signal (WRB)**
  - když WRB = 1, tak zapiš data z DI do registru s adresou ADI
- **Constant Values**
  - v sadě registrů jsou trvale uloženy konstanty
  - 11111111=FFh, 00000000=00h
  - využívají se pro některé operace

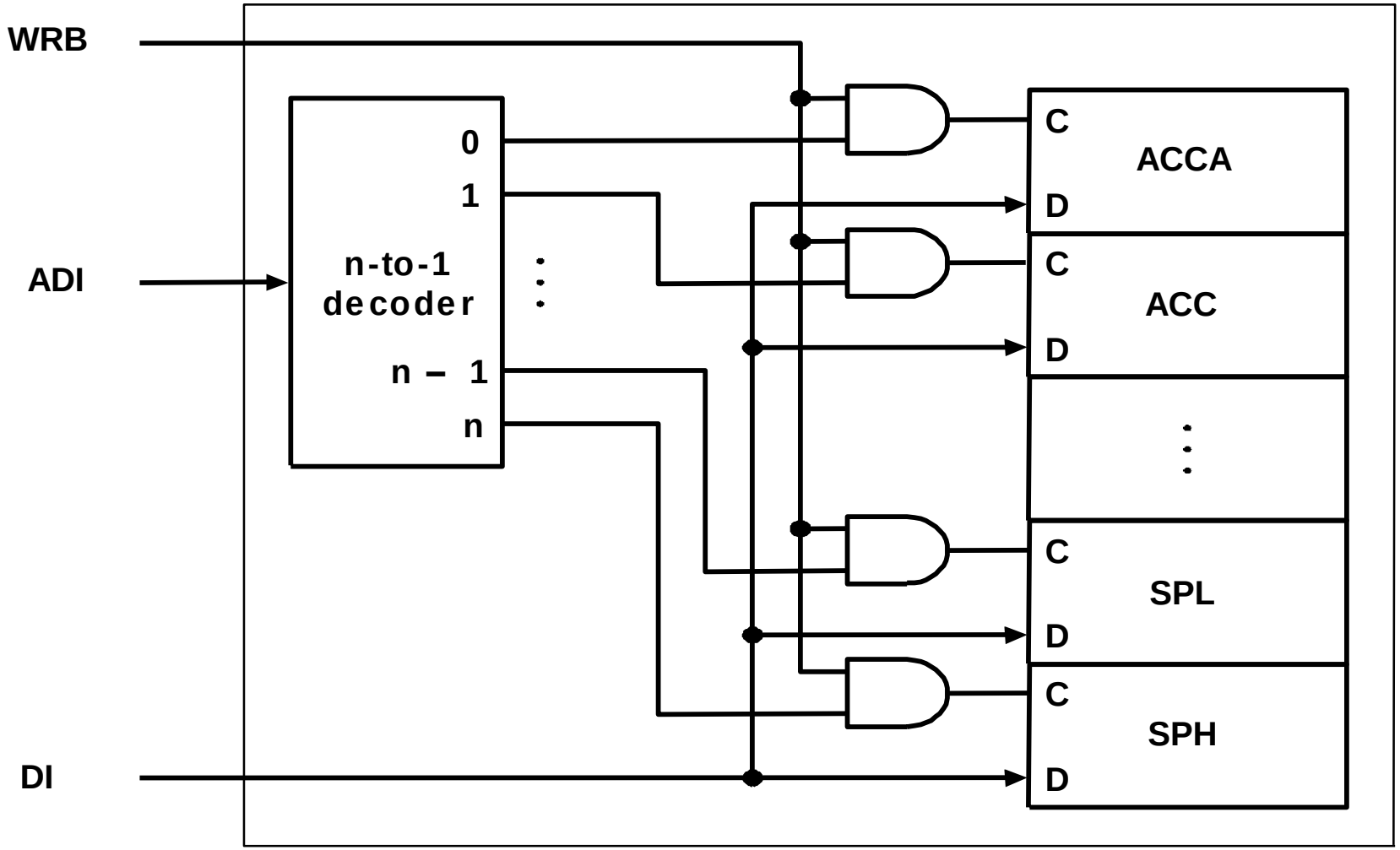


HC11: Register Bank

- Čte se ze dvou registrů naráz

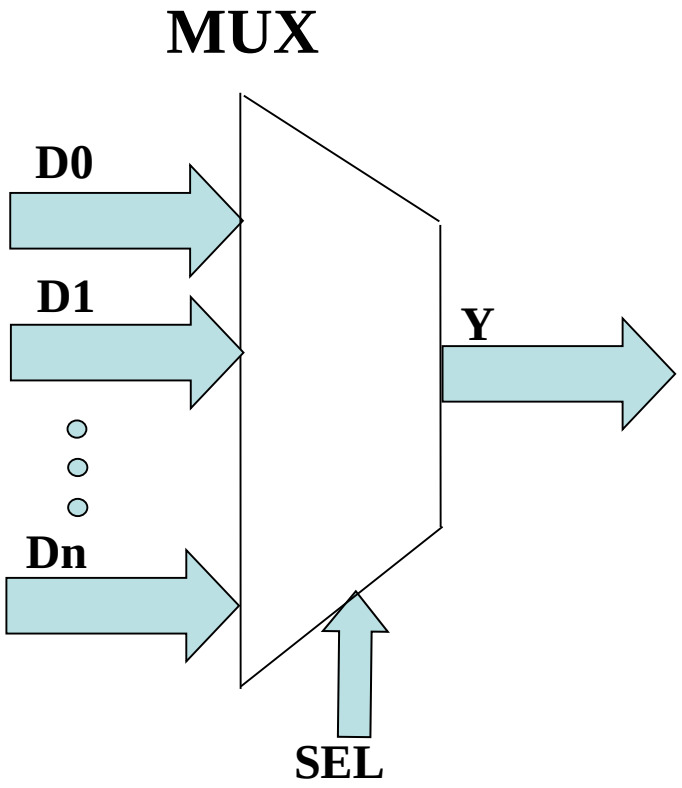


- Zapisuje se do jednoho registru



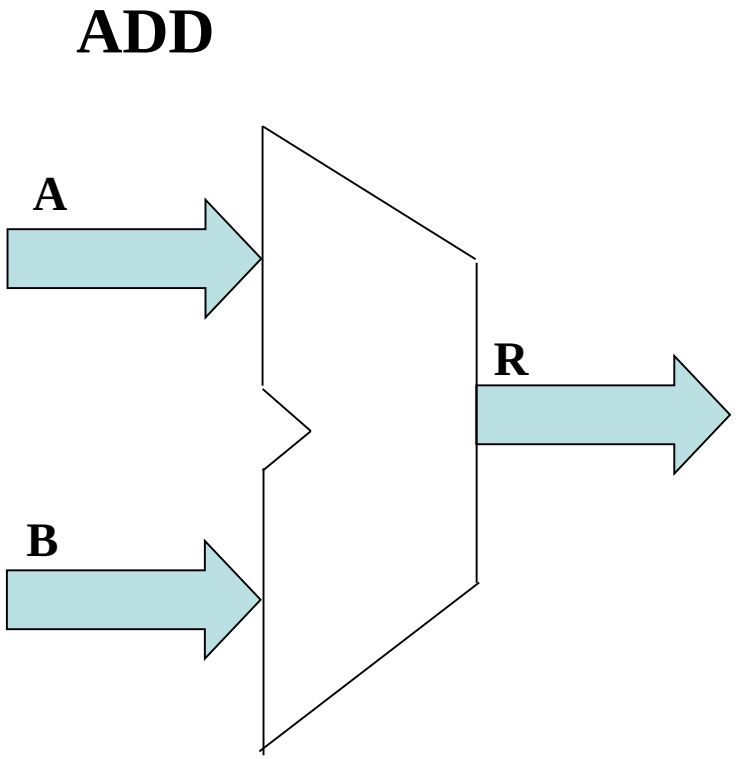
HC11: Multiplexor

- N vstupních datových sběrnic (2 .. N)
  - Využívají se pouze dvou a tří vstupové MUXy
  - Šířka 1 až 16 bitů
- Výběr vstupní sběrnice (SEL)
  - Vybírá jednu ze vstupních sběrnic na výstup Y
- Výstupní sběrnice (Y)
  - Má stejnou šířku jako vstupní sběrnice



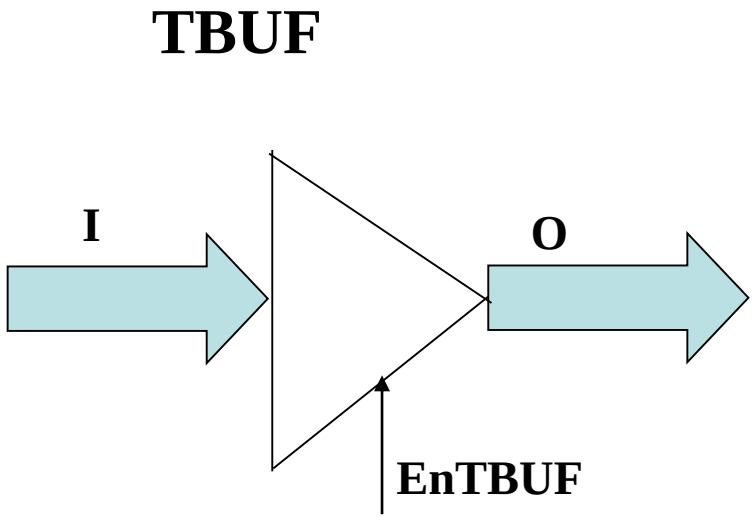
HC11: Sčítačka

- Použití
  - Inkrementace programového čítače (Program Counter - PC) o jedničku
  - Přičtení ofsetu k současné hodnotě PC v případě vykonávání instrukce skoku (Branch)
- Přičítá 8-mi bitová čísla ve dvojkovém doplňku
  - Skok dopředu – přičtení kladného čísla
  - Skok dozadu – přičtení záporného čísla



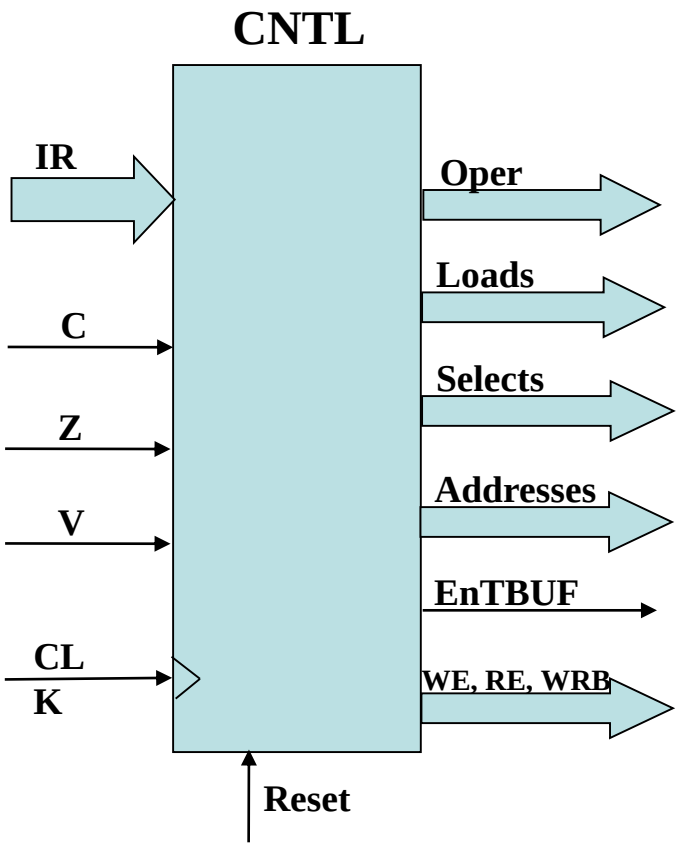
HC11: Třístavový budič

- Tristate Buffer (TBUF)
  - Když E = 1, tak O = I jinak O = Z
  - Z ... High Impedance (odpojeno)
- Použití
  - Pro zápis do paměti, která má v sobě též třístavové budiče
- Zápis do paměti
  - TBUF je otevřen
- Čtení z paměti
  - TBUF zavřen a výstupní budič v paměti je otevřen



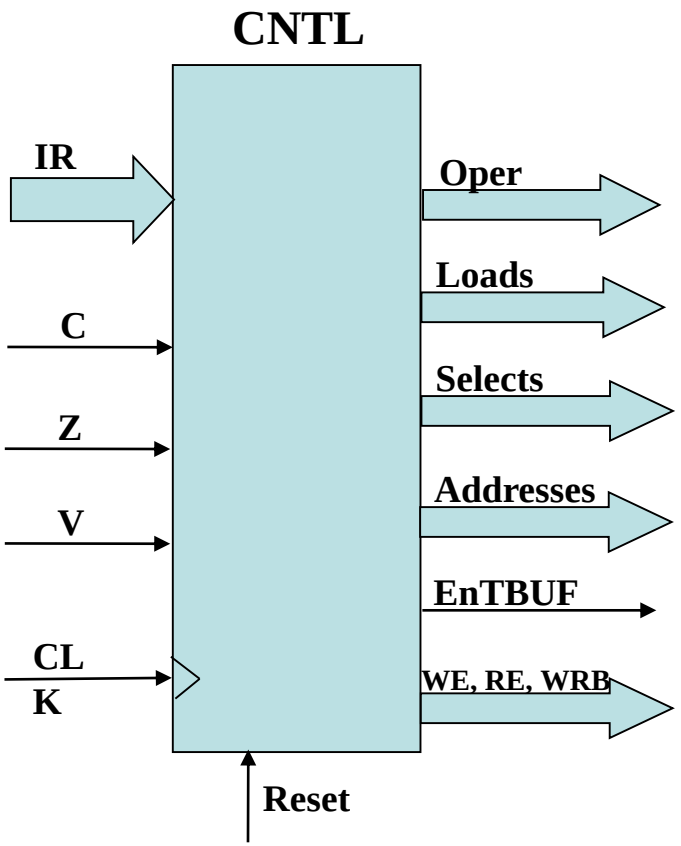
HC11: Řadič

- „Srdce“ HC11
  - Na základě operačního kódu instrukce (OpCode) přečtené z paměti a stavu příznakových registrů generuje posloupnost řídicích signálů, které ovládají činnost jednotlivých bloků procesoru
- Vstupy
  - Výstup IR registru – současný OpCode
  - Příznaky C, Z a V
  - Hodiny - CLK
  - Reset



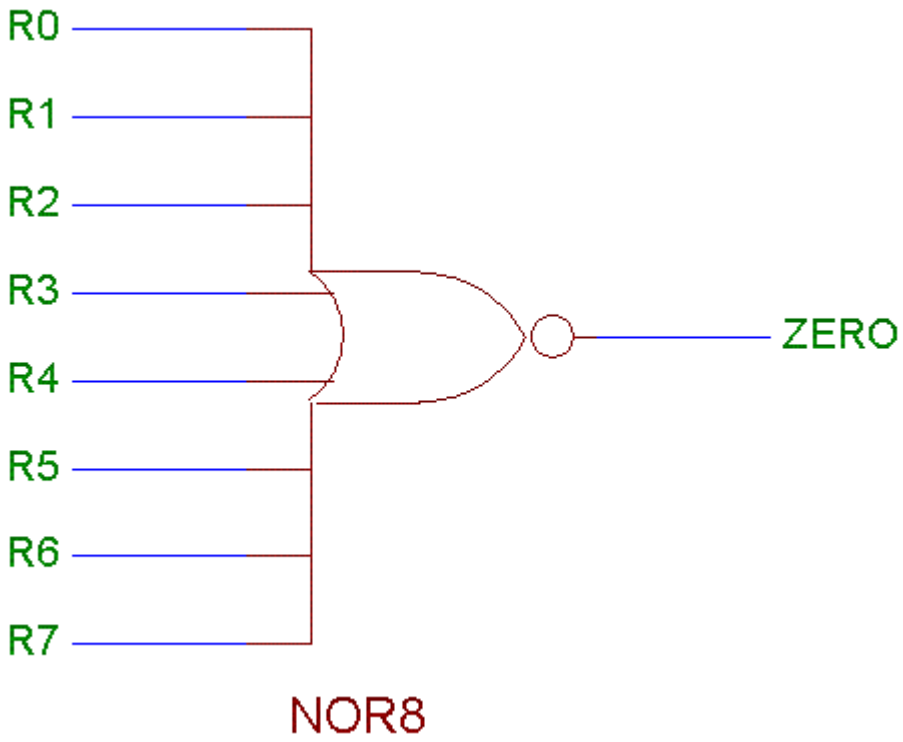
HC11: Řadič

- Oper - ALU operace (ADD, SUB, AND...)
- Load (ulož do registru) signály
  - LdIR, LdPC, LdAL, LdAH, LdM, LdC, LdZ, LdV, LdTC a LdR
- Select (výběr vstupu MUXu) signals
  - SelPC, SelAL, SelAH, SelMW, SelADDR, SelB, SelC, SelCIN a SelV
- Adresy do sady registrů (Register Bank)
  - ADI, ADA a ADB
- Výstupní budič Enable (zápis do paměti)
  - EnTBUF
- Zápis do sady registrů (Register Bank)
  - WRB
- Zápis/čtení z paměti
  - WE, RE

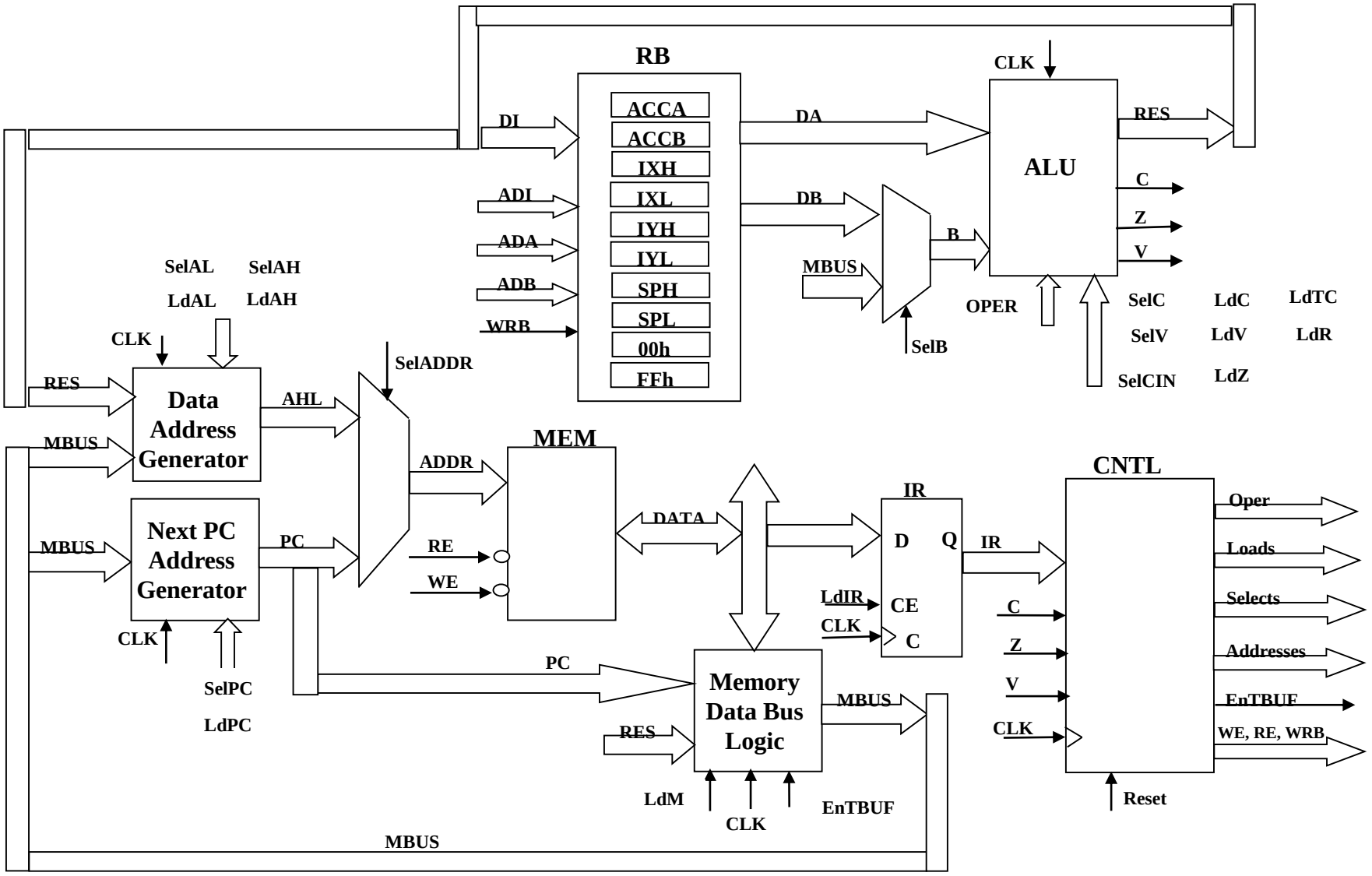


HC11: Test na nulu

- Test, zda je výsledek operace roven nule
  - N-bitový NOR

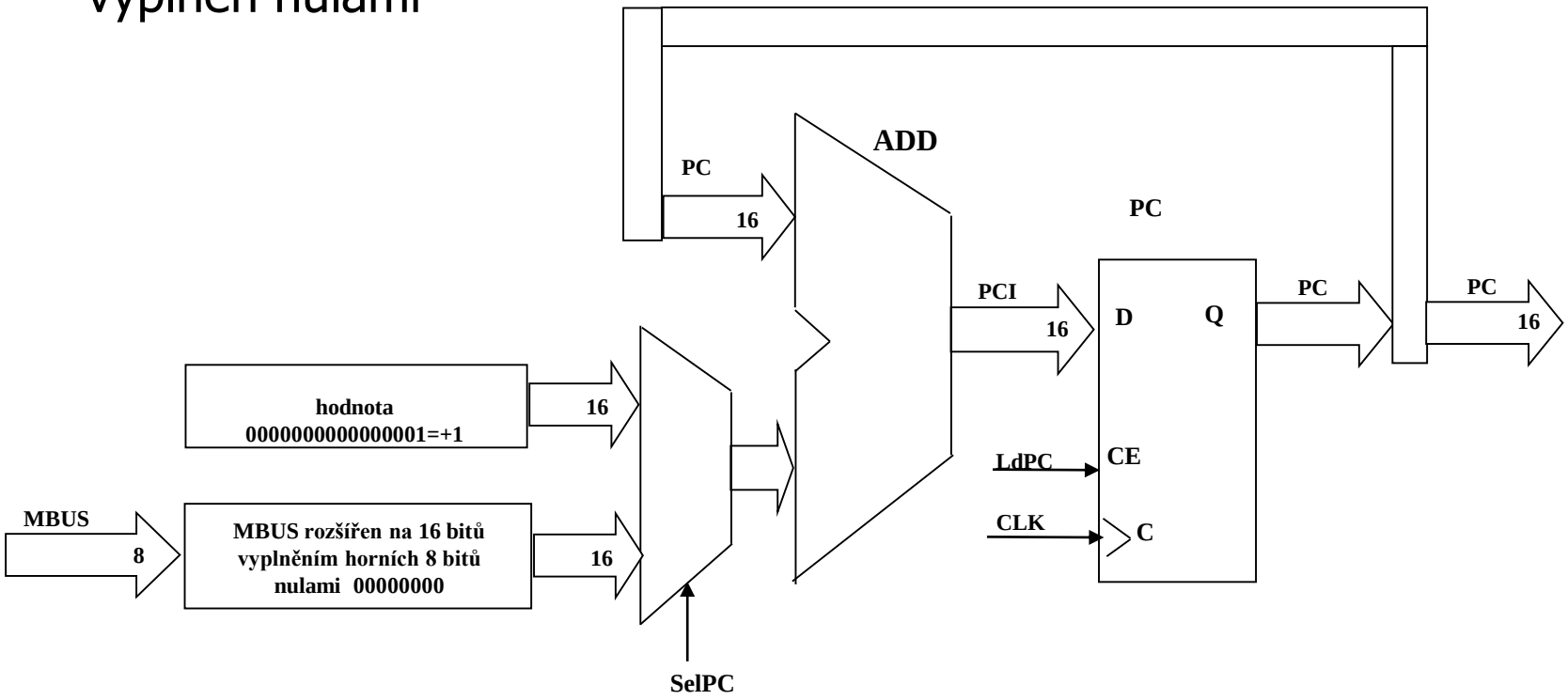


HC11: Datová cesta



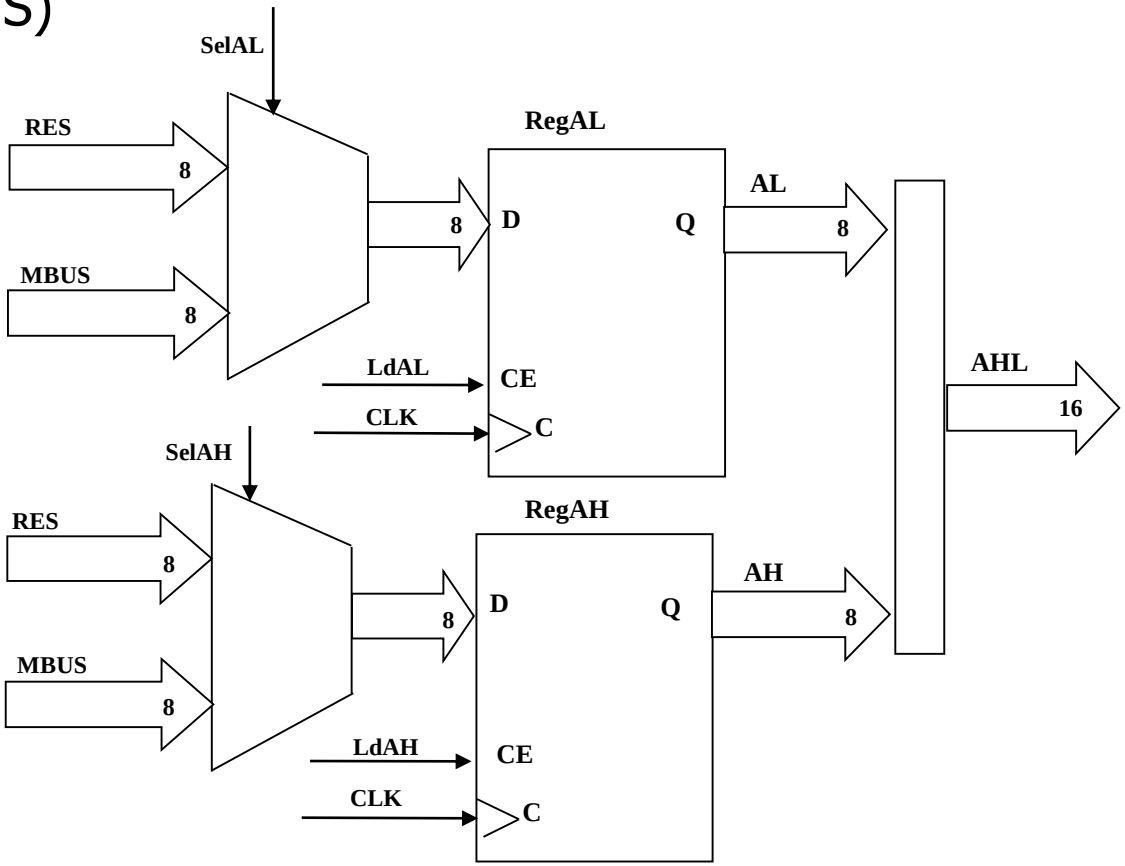
HC11: Generátor následující adresy PC

- Inkrementuj PC – sekvenční vykonávání programu
  - Přičti hodnotu +1 k PC
- Výpočet ofsetu skoku (Branch) – větvení programu
  - Přičti 8-bitovou hodnotu (offset) přečtenou z paměti k PC
  - Protože je ofset jen 8-bitový a sčítačka 16-bitová, je horní byte vyplněn nulami



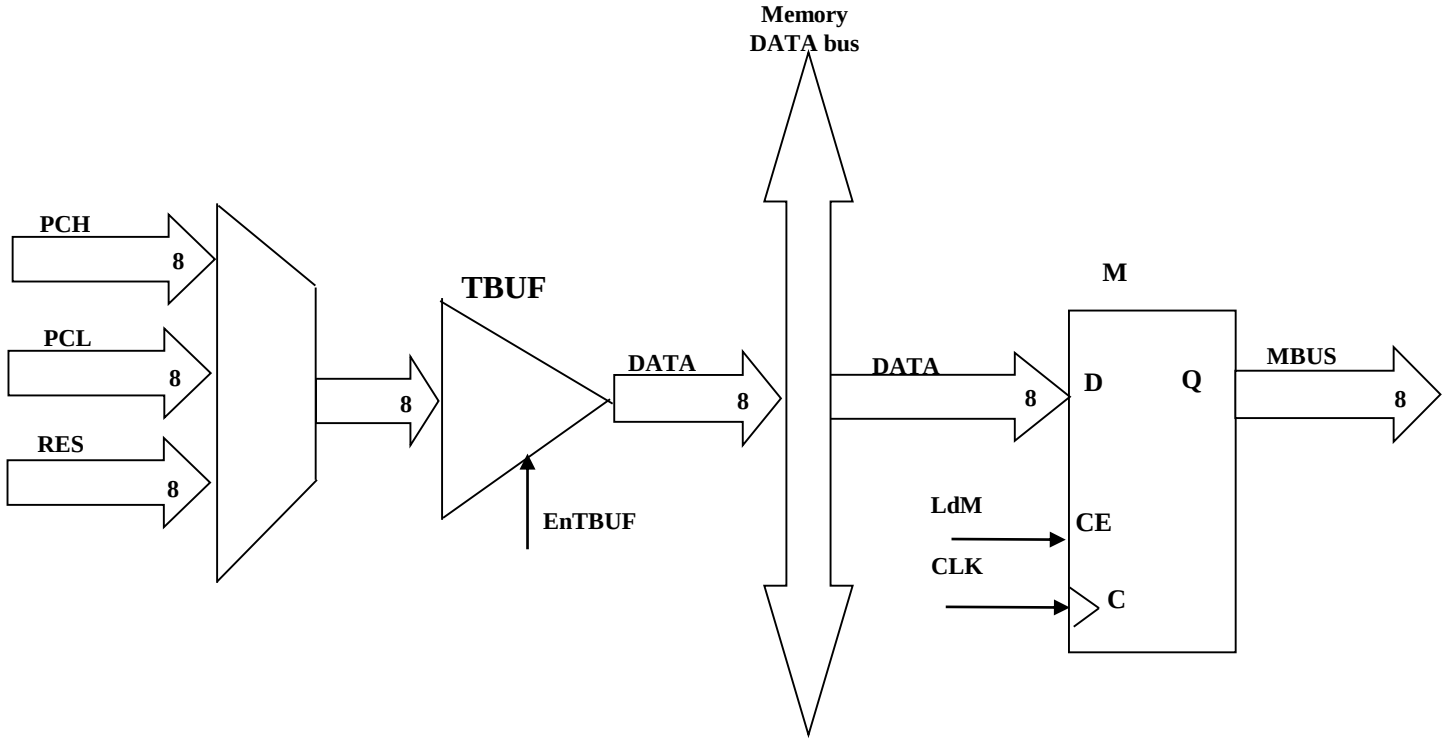
HC11: Generátor ukazatele (adresy) dat v paměti

- Adresa dat (ukazatel na data)
  - Indexed addressing mode - adresa je vypočtena v ALU (sběrnice RES)
  - Direct and extended addressing modes – adresa je čtena z paměti (MBUS)



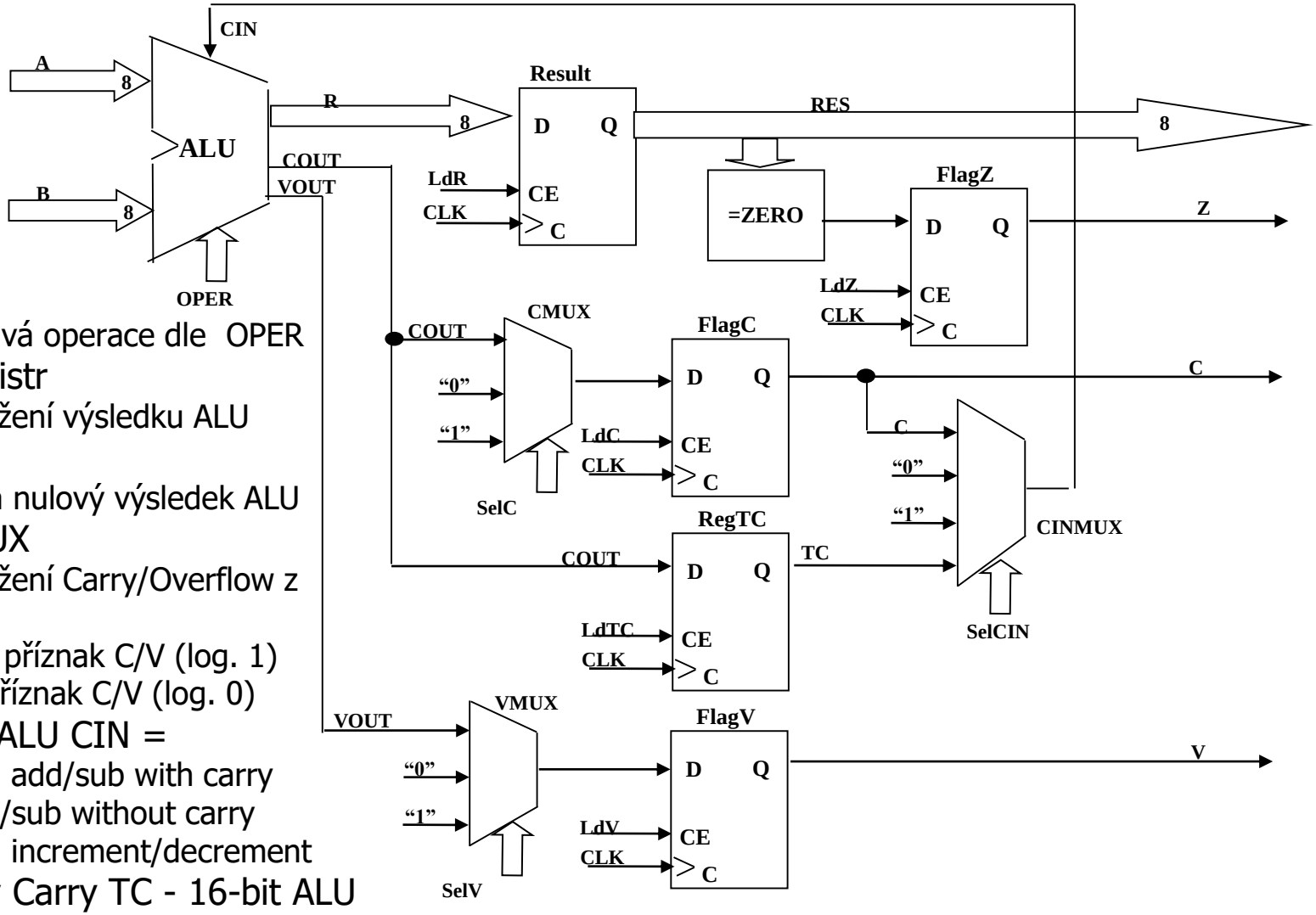
HC11: Zdroje dat a zápis do paměti

- M Registr
  - Je určen pro uložení dat či OpCode
- TBUF – třístavový výstupní budič pro zápis do paměti
  - Výsledek ALU (sběrnice RES)
  - Uložení obsahu PC do paměti (sběrnice PCL a PCH) – návratová adresa při volání podprogramu či obsluze přerušení

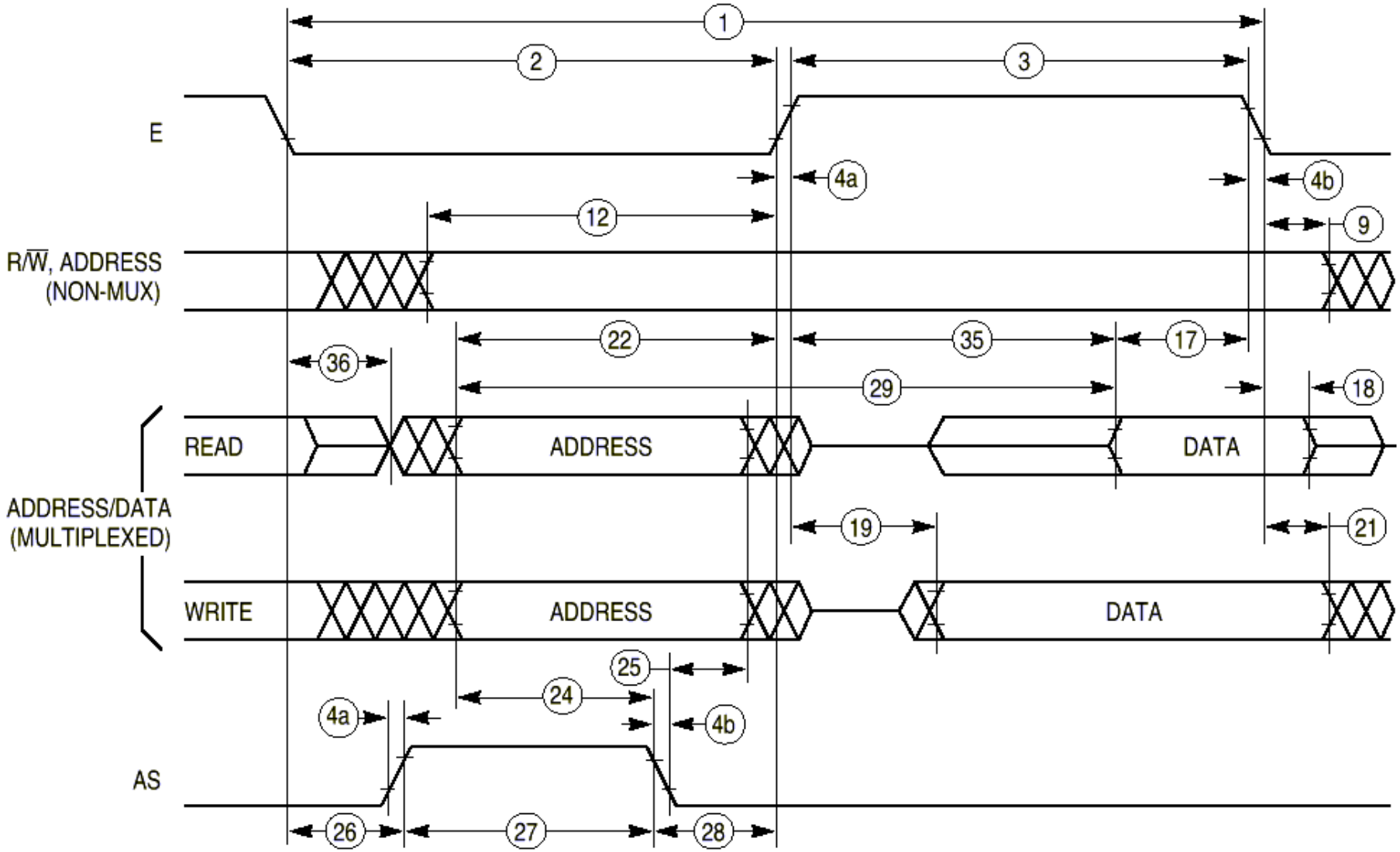


HC11: ALU a příznaky (Flags)

- ALU
  - Vykonává operace dle OPER
- Result Registr
  - Pro uložení výsledku ALU
- ZERO
  - Test na nulový výsledek ALU
- CMUX/VMUX
  - Pro uložení Carry/Overflow z ALU
  - Nastav příznak C/V (log. 1)
  - Nuluj příznak C/V (log. 0)
- CINMUX - ALU CIN =
  - C flag - add/sub with carry
  - 0 - add/sub without carry
  - 1 - e.g. increment/decrement
- Temporary Carry TC - 16-bit ALU operations
  - C, Z, V příznaky jsou součástí CCR



HC11: Časový diagram paměťové sběrnice



HC11: Příklad instrukce – ulož do akumulátoru

LDA Load Accumulator LDA

Operation: ACCX ⇐ (M)

Description: Loads the contents of memory into the 8-bit accumulator. The condition codes are set according to the data.

Condition Codes and Boolean Formulae:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	0	—

N R7
Set if MSB of result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if result is $00; cleared otherwise.

V 0
Cleared

Source Forms: LDAA (opr); LDAB (opr)

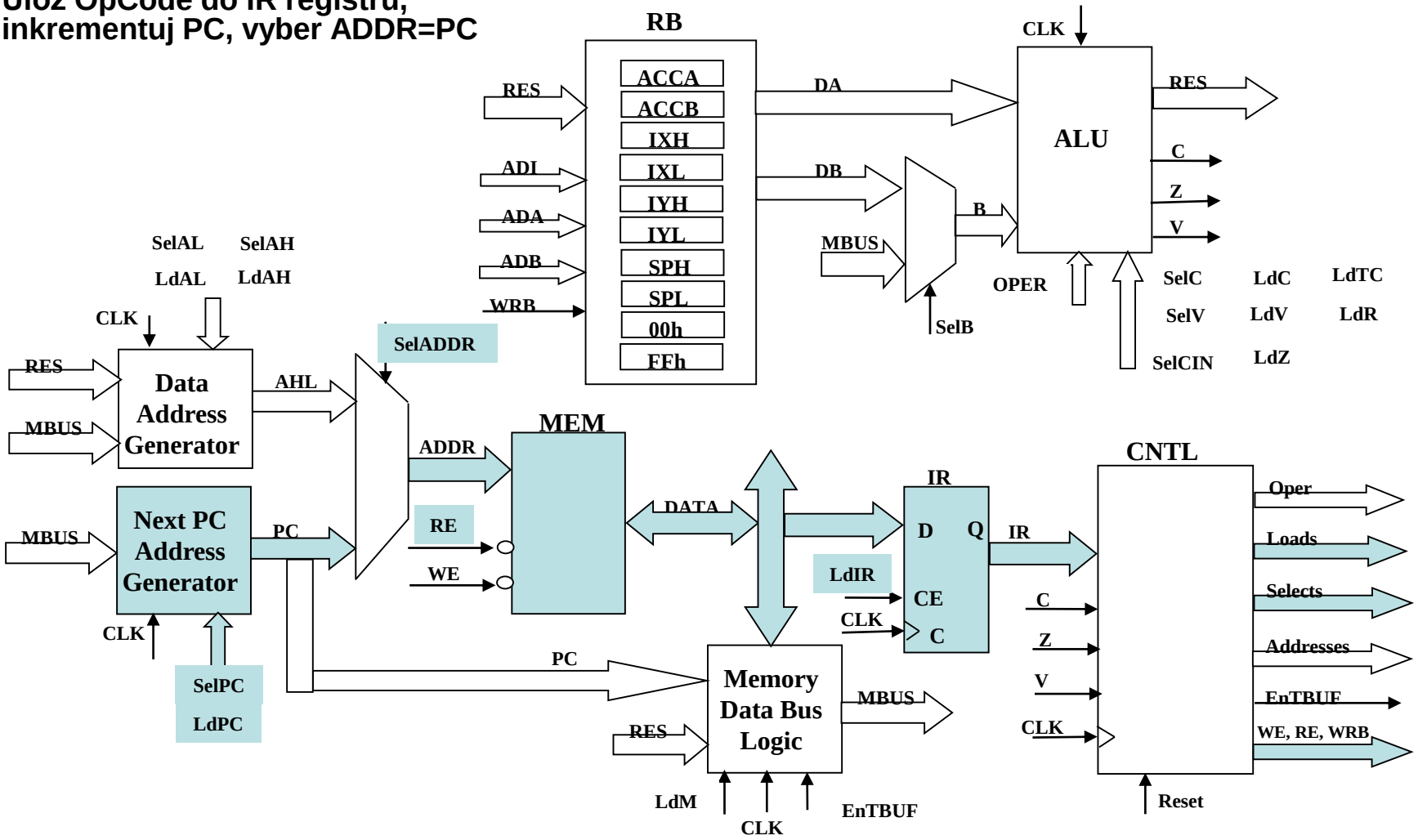
Addressing Modes, Machine Code, and Cycle-by-Cycle Execution:

Cycle	LDAA (IMM)			LDAA (DIR)			LDAA (EXT)			LDAA (IND,X)			LDAA (IND,Y)		
	Addr	Data	R/W	Addr	Data	R/W	Addr	Data	R/W	Addr	Data	R/W	Addr	Data	R/W
1	OP	86	1	OP	96	1	OP	B6	1	OP	A6	1	OP	18	1
2	OP+1	ii	1	OP+1	dd	1	OP+1	hh	1	OP+1	ff	1	OP+1	A6	1
3				00dd	(00dd)	1	OP+2	ll	1	FFFF	—	1	OP+2	ff	1
4							hhll	(hhll)	1	X+ff	(X+ff)	1	FFFF	—	1
5													Y+ff	(Y+ff)	1

Cycle	LDAB (IMM)			LDAB (DIR)			LDAB (EXT)			LDAB (IND,X)			LDAB (IND,Y)		
	Addr	Data	R/W	Addr	Data	R/W	Addr	Data	R/W	Addr	Data	R/W	Addr	Data	R/W
1	OP	C6	1	OP	D6	1	OP	F6	1	OP	E6	1	OP	18	1
2	OP+1	ii	1	OP+1	dd	1	OP+1	hh	1	OP+1	ff	1	OP+1	E6	1
3				00dd	(00dd)	1	OP+2	ll	1	FFFF	—	1	OP+2	$\overline{ff}$	1
4							hhll	(hhll)	1	X+ff	(X+ff)	1	FFFF	—	1
5													Y+ff	(Y+ff)	1

E Clock 1/fáze 1

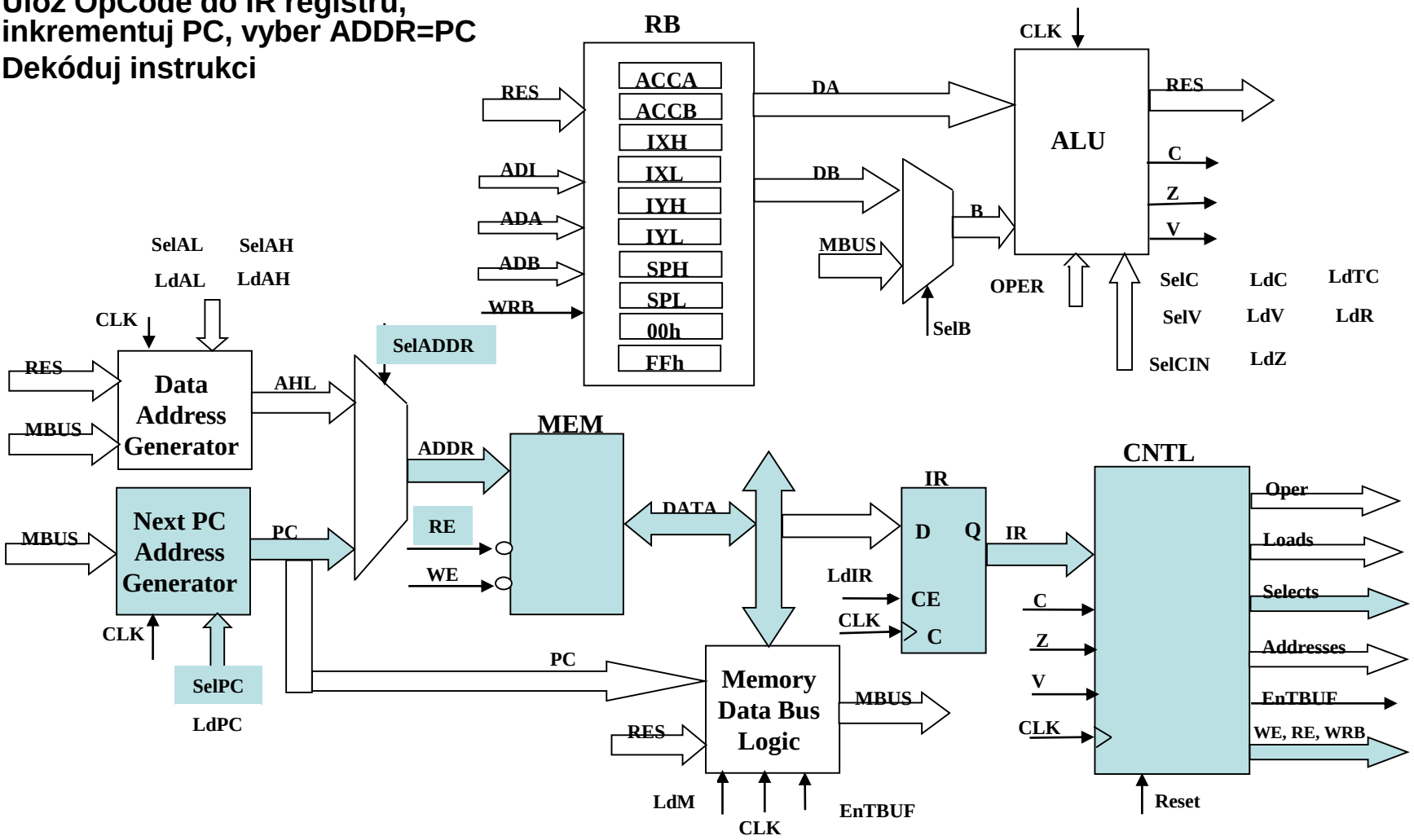
Ulož OpCode do IR registru, inkrementuj PC, vyber ADDR=PC



E Clock 1/fáze 2

Ulož OpCode do IR registru, inkrementuj PC, vyber ADDR=PC

Dekóduj instrukci

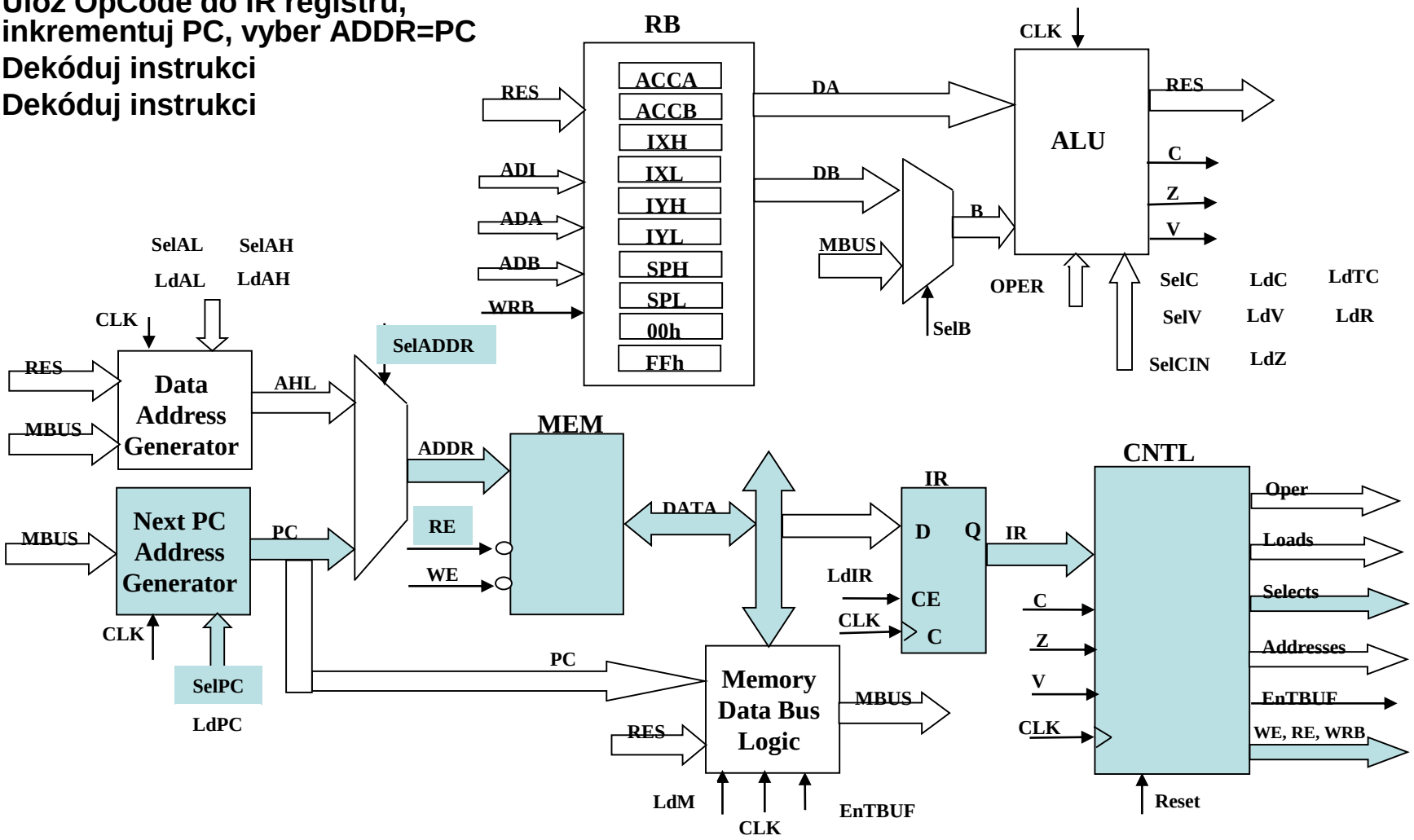


E Clock 1/fáze 3

Ulož OpCode do IR registru, inkrementuj PC, vyber ADDR=PC

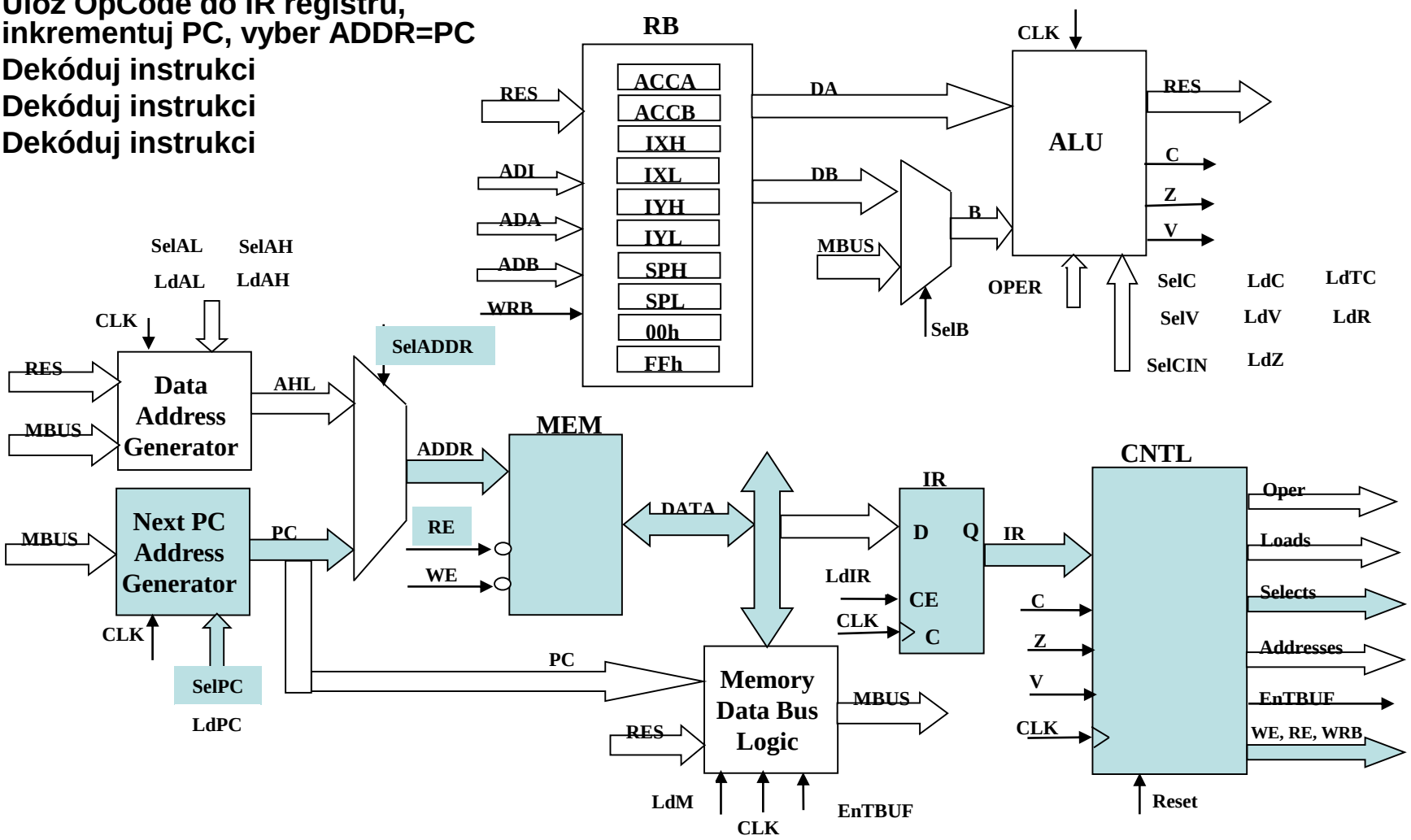
Dekóduj instrukci

Dekóduj instrukci



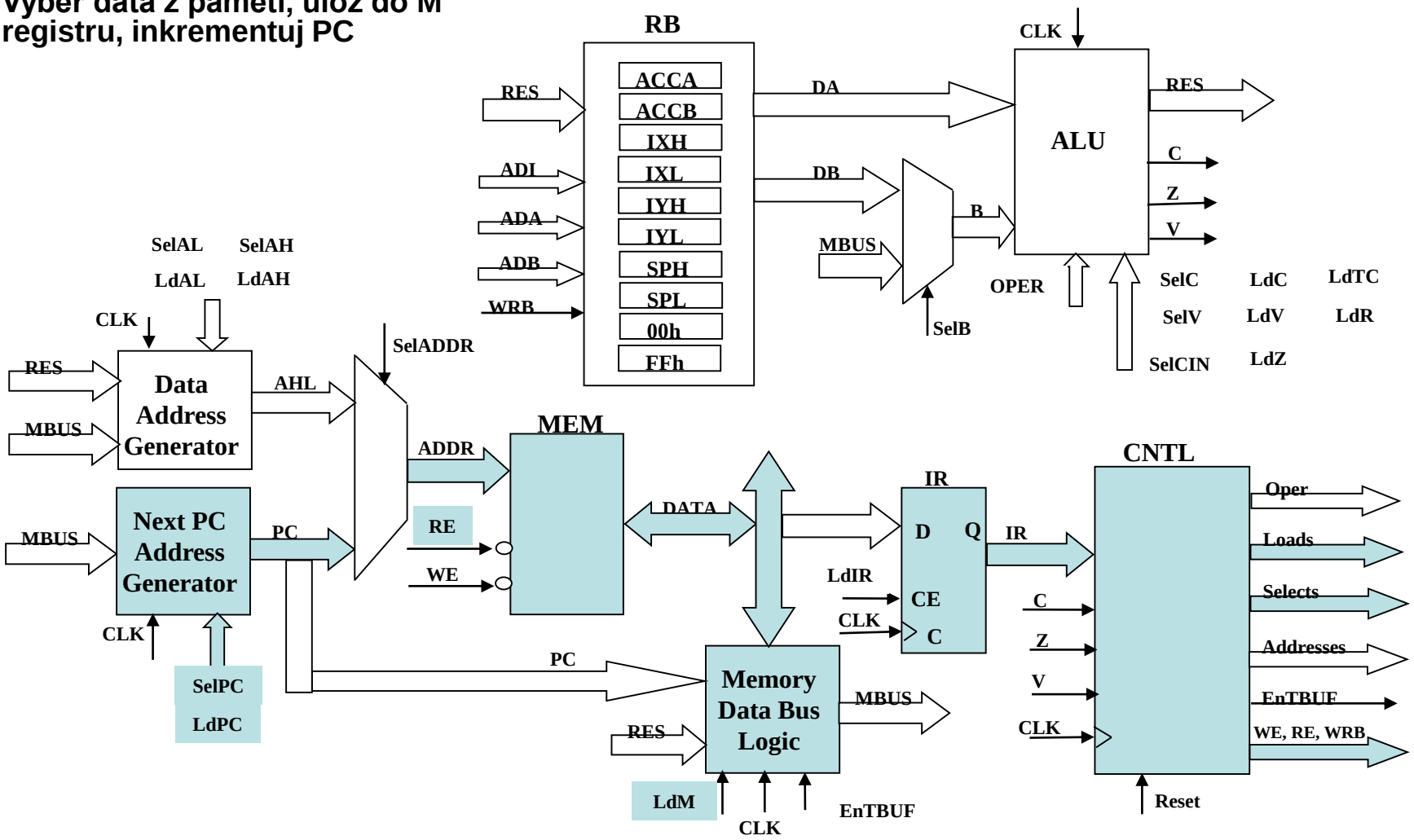
E Clock 1/fáze 4

Ulož OpCode do IR registru, inkrementuj PC, vyber ADDR=PC
Dekóduj instrukci
Dekóduj instrukci
Dekóduj instrukci



E Clock 2/fáze 1

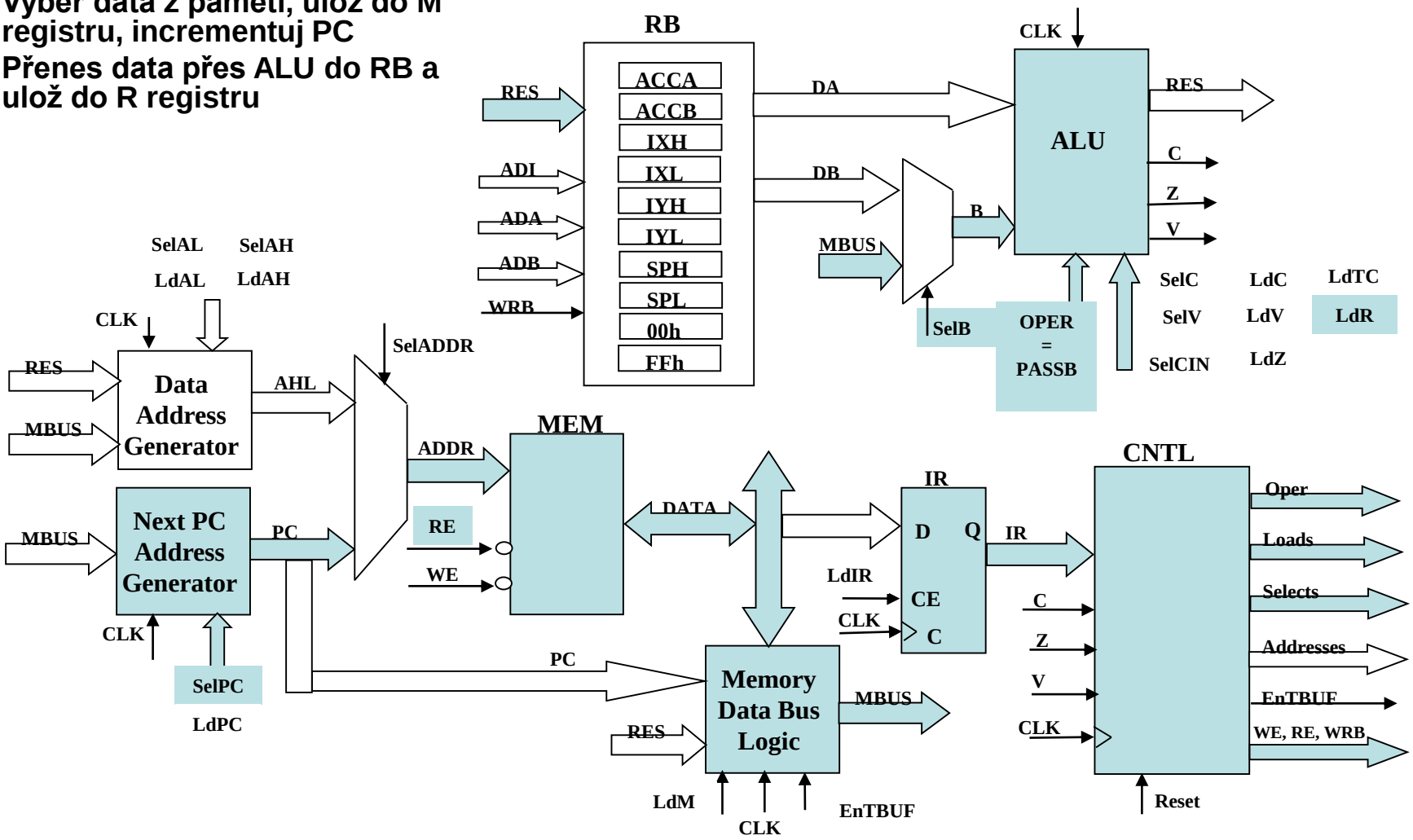
Vyber data z paměti, ulož do M registru, inkrementuj PC



E Clock 2/fáze 2

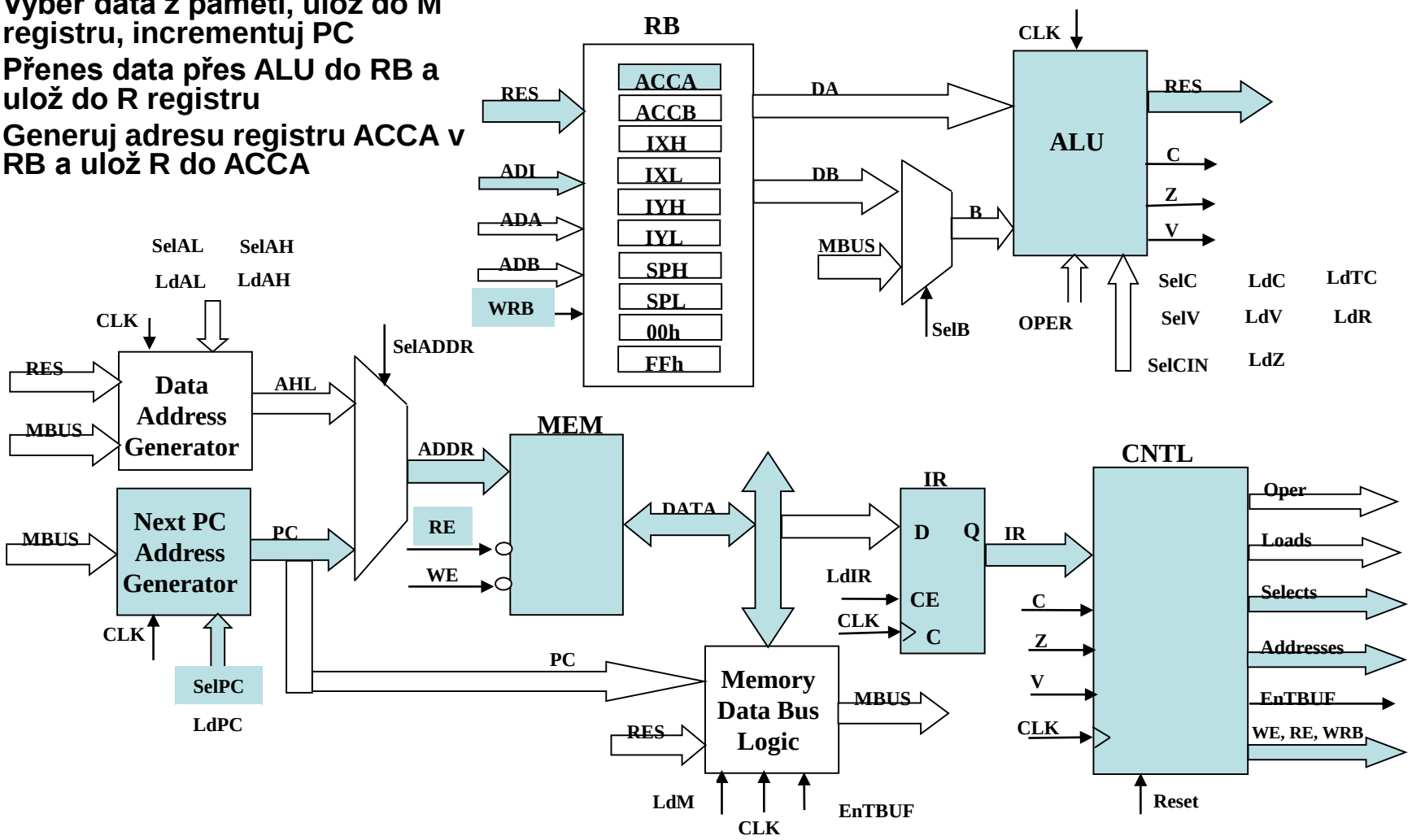
Vyber data z paměti, ulož do M registru, incrementuj PC

Přenes data přes ALU do RB a ulož do R registru



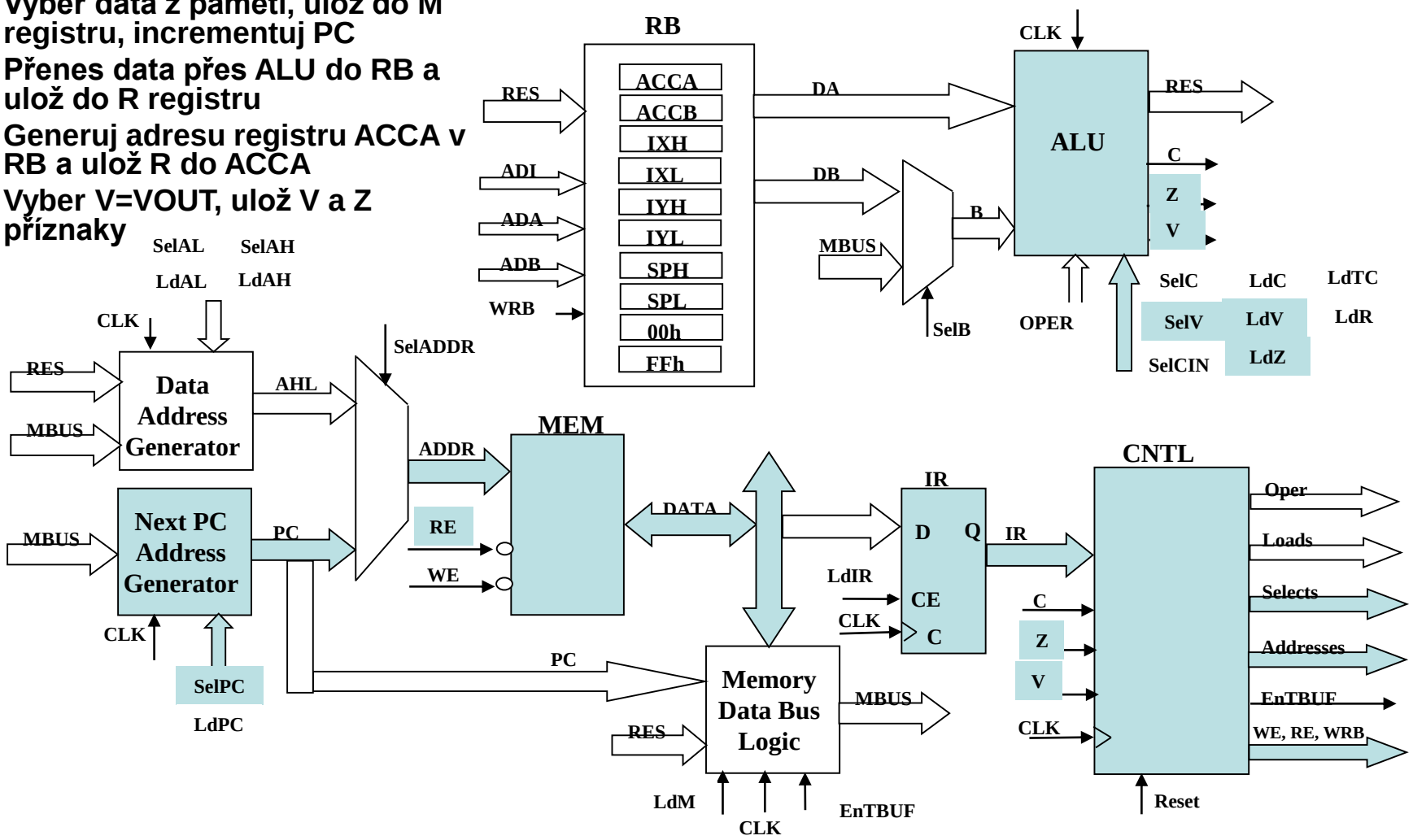
E Clock 2/fáze 3

- Vyber data z paměti, ulož do M registru, incrementuj PC
- Přenes data přes ALU do RB a ulož do R registru
- Generuj adresu registru ACCA v RB a ulož R do ACCA



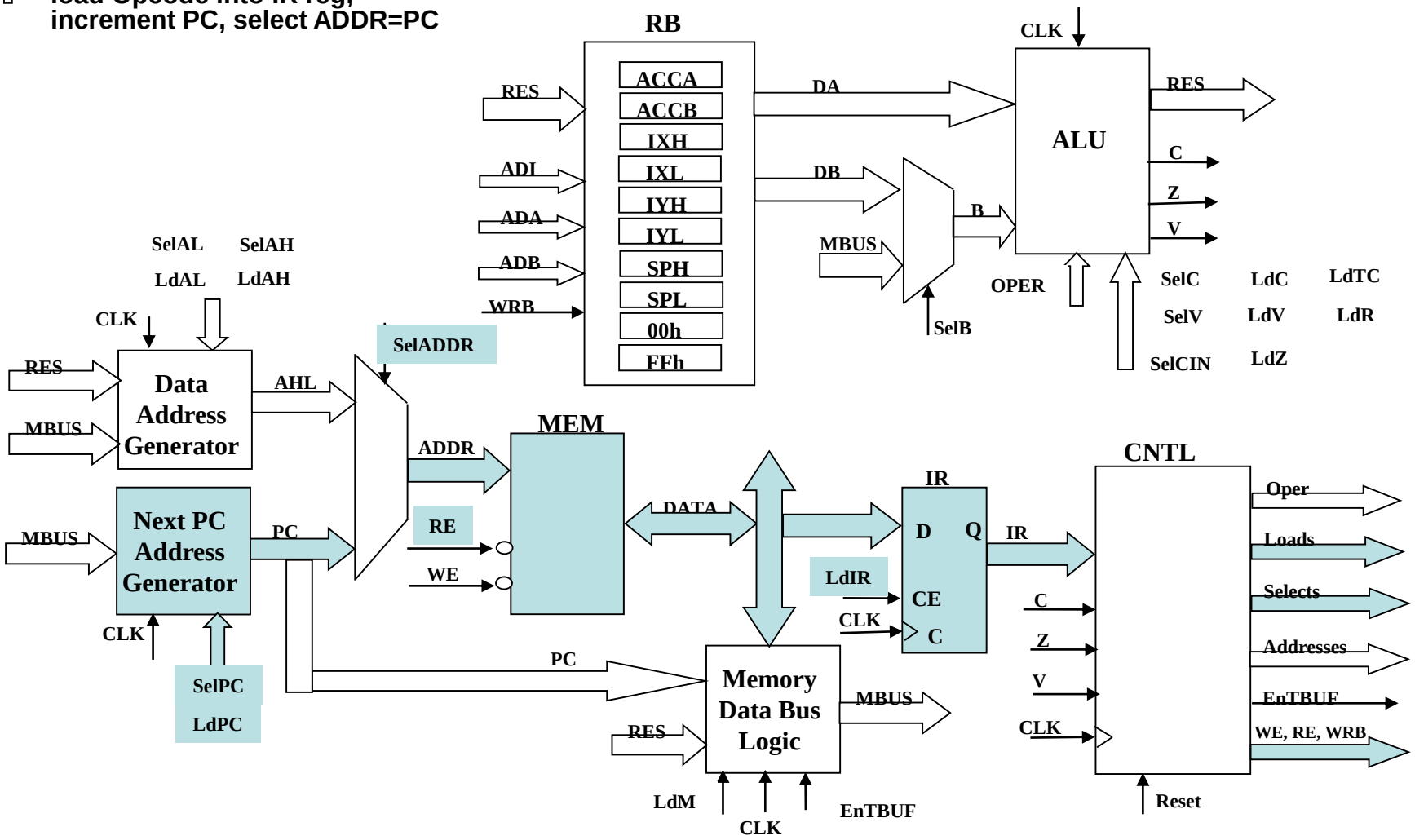
E Clock 2/fáze 4

- Vyber data z paměti, ulož do M registru, incrementuj PC
- Přenes data přes ALU do RB a ulož do R registru
- Generuj adresu registru ACCA v RB a ulož R do ACCA
- Vyber V=VOUT, ulož V a Z příznaky

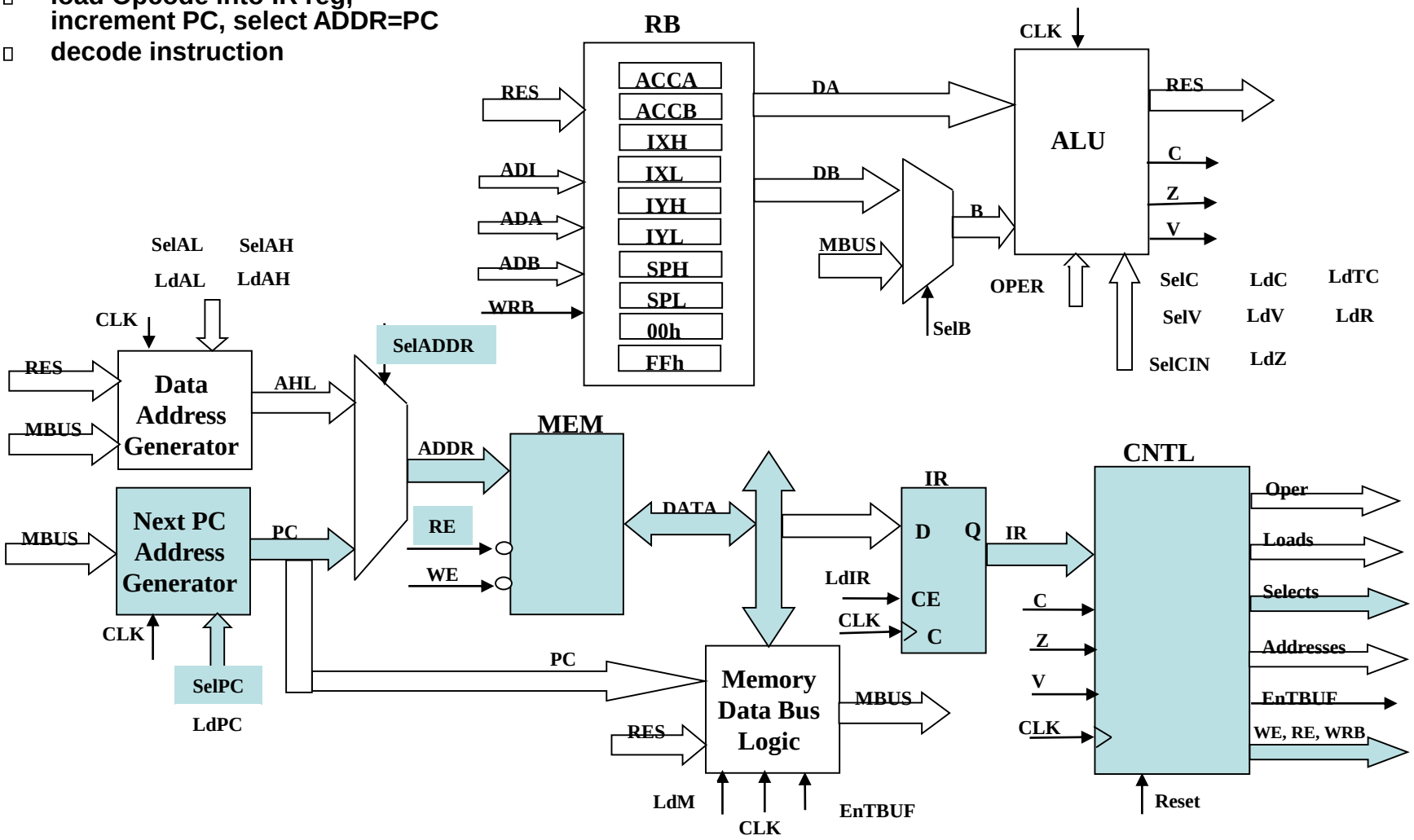


HC11: Instrukce LDAB, adresovací režim IND, X

- **E Clock #1**
  - **load Opcode into IR reg, increment PC, select ADDR=PC**

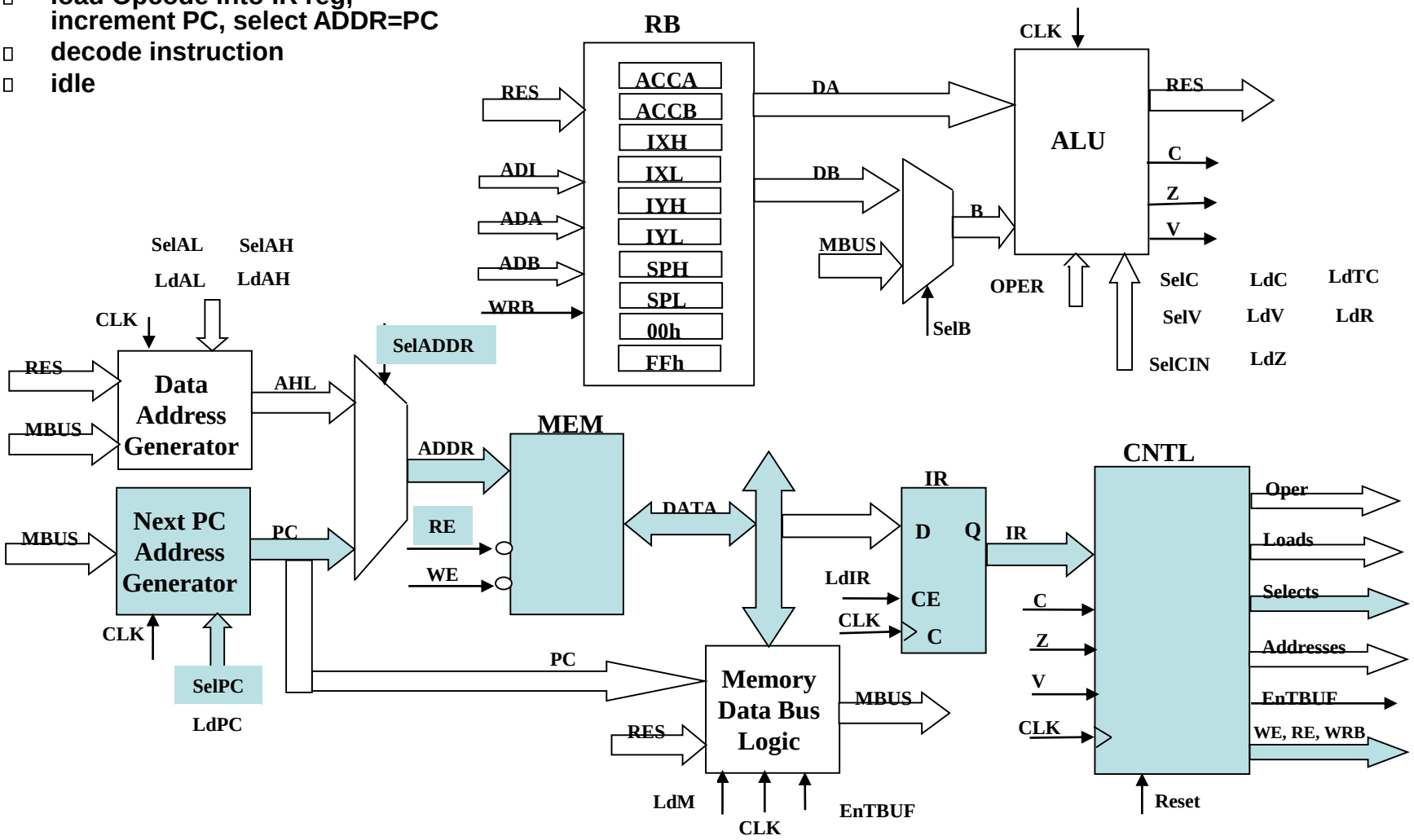


- **E Clock #1**
  - load Opcode into IR reg, increment PC, select ADDR=PC
  - decode instruction



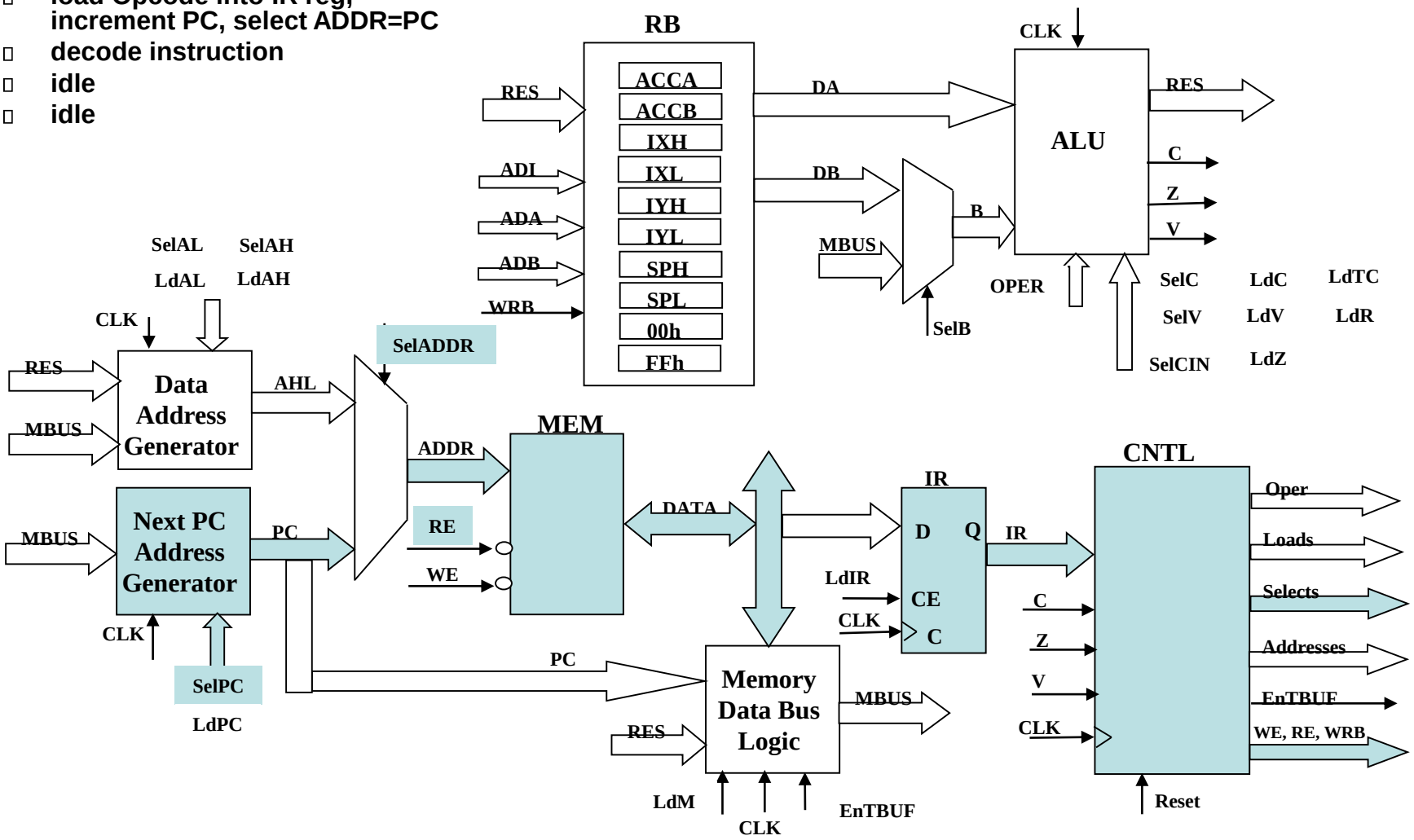
HC11: Instrukce LDAB, adresovací režim IND, X

- **E Clock #1**
  - load Opcode into IR reg, increment PC, select ADDR=PC
  - decode instruction
  - idle

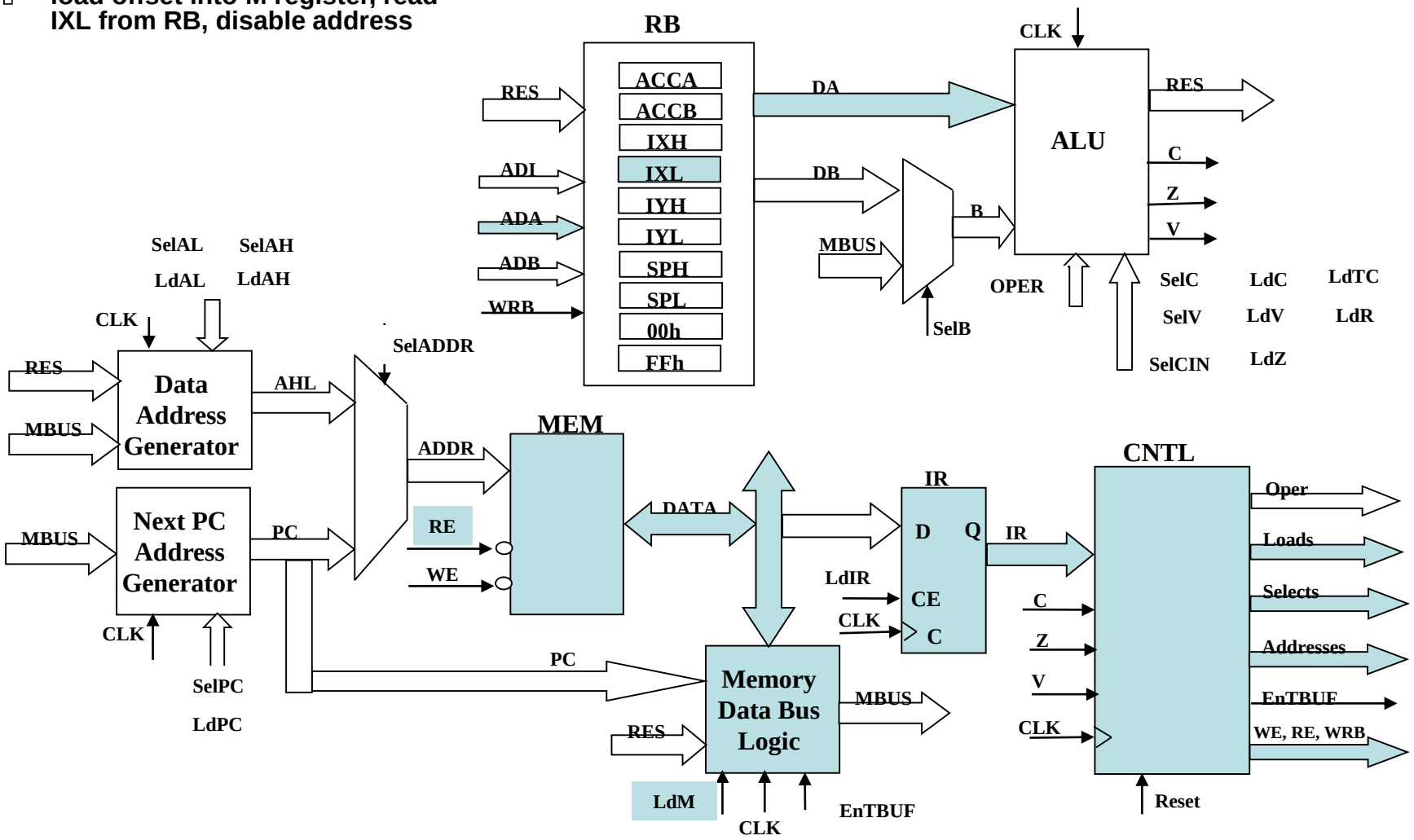


HC11: Instrukce LDAB, adresovací režim IND, X

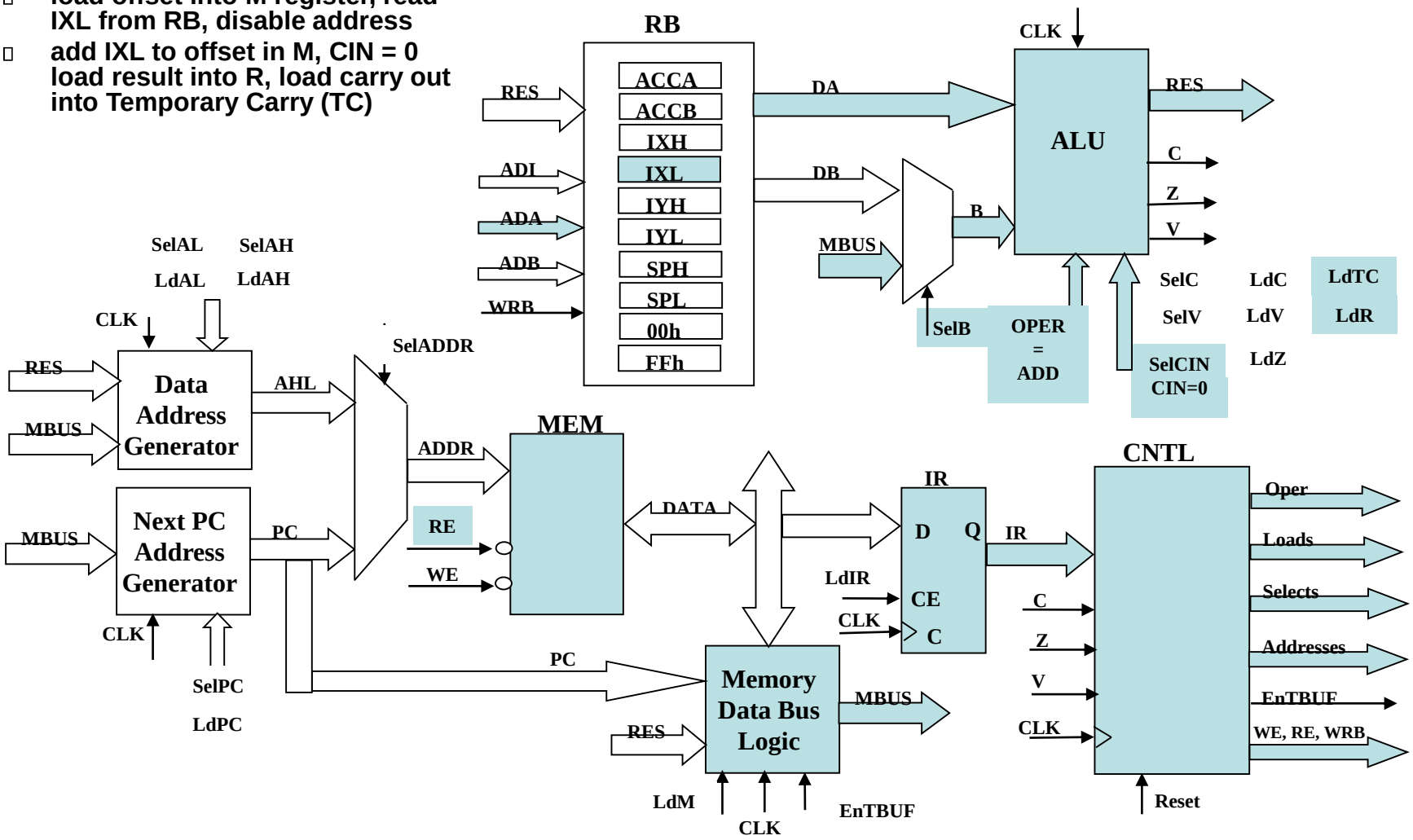
- **E Clock #1**
  - load Opcode into IR reg, increment PC, select ADDR=PC
  - decode instruction
  - idle
  - idle



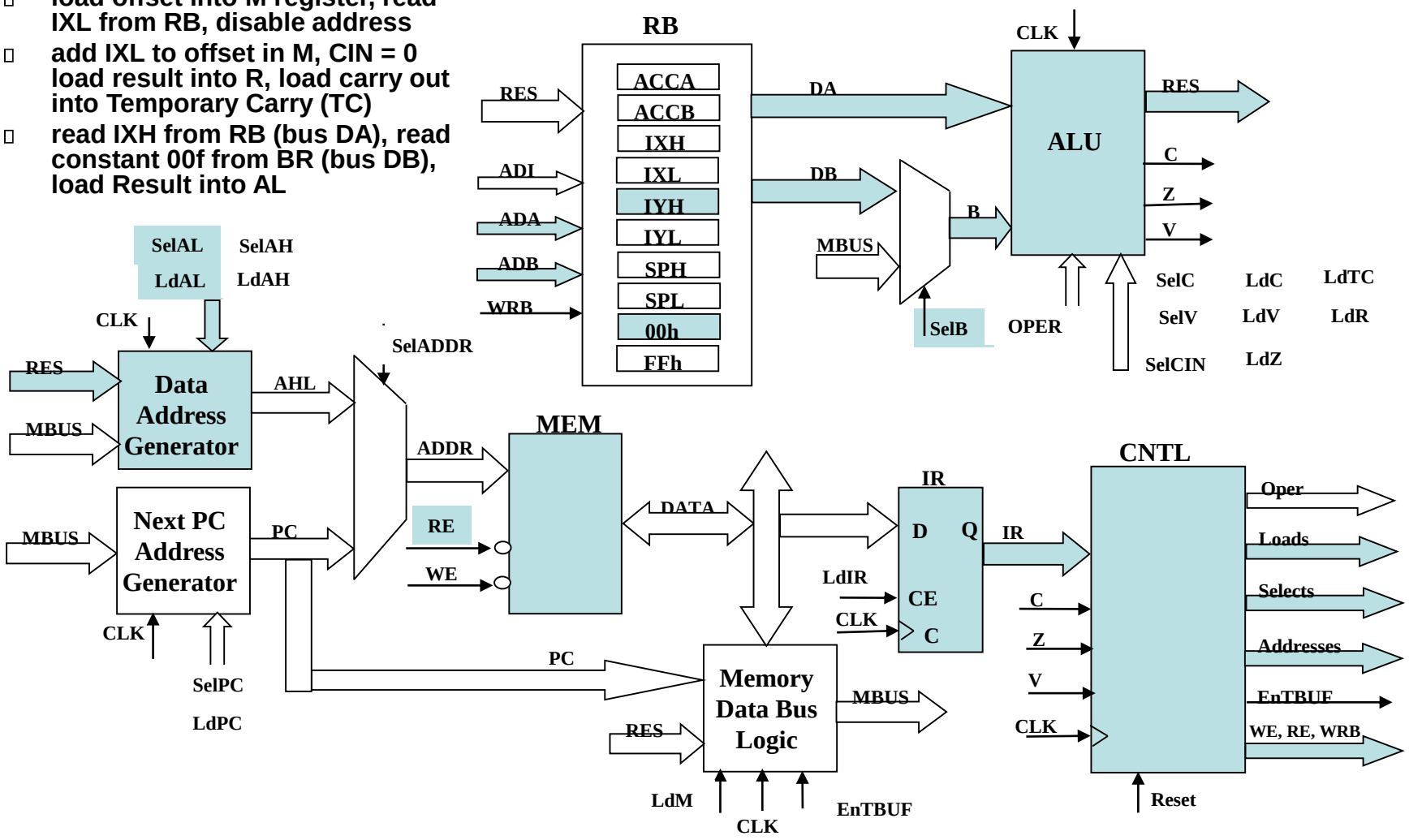
- **E Clock #2**
  - **load offset into M register, read IXL from RB, disable address**



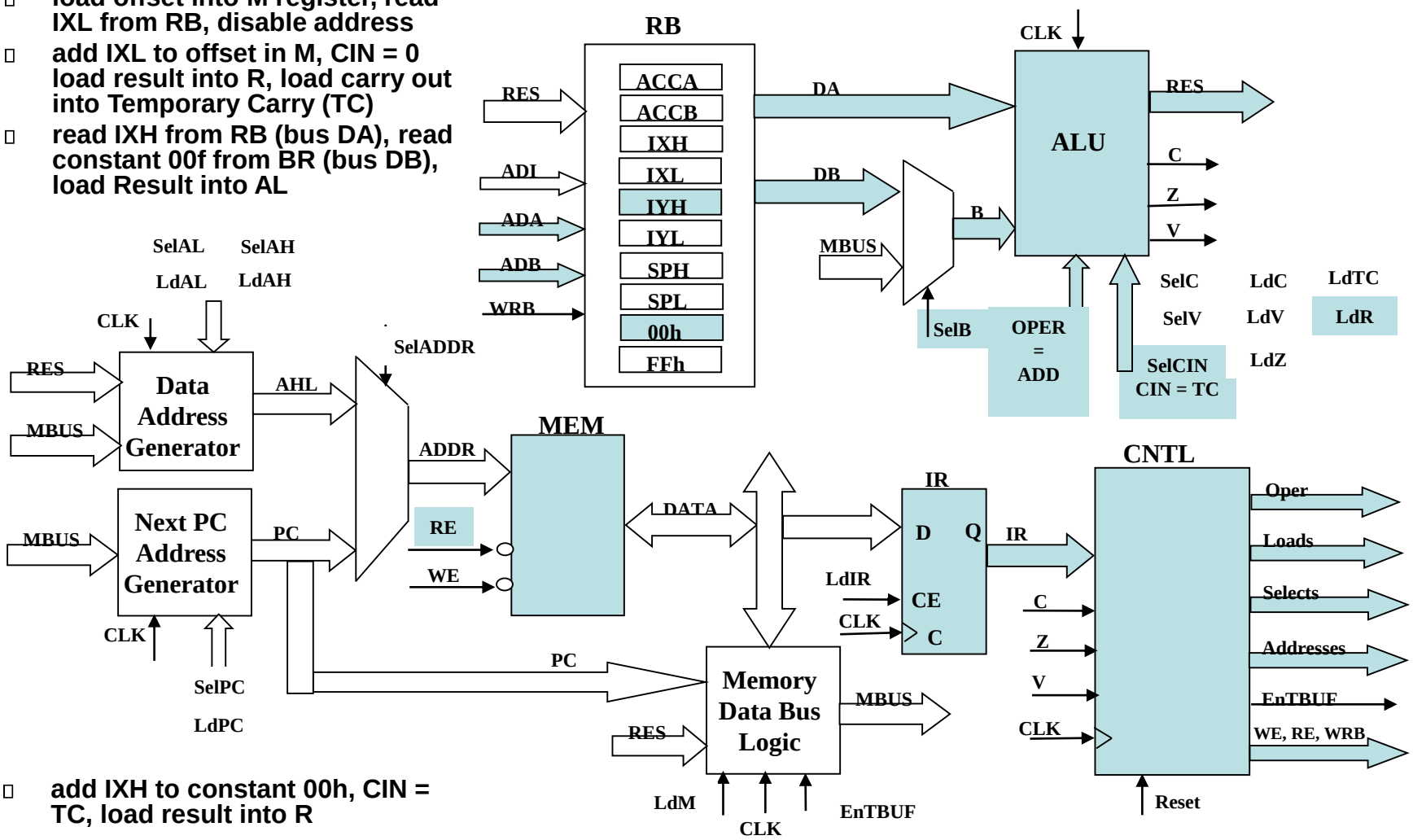
- **E Clock #2**
  - load offset into M register, read IXL from RB, disable address
  - add IXL to offset in M, CIN = 0 load result into R, load carry out into Temporary Carry (TC)



- **E Clock #2**
  - load offset into M register, read IXL from RB, disable address
  - add IXL to offset in M, CIN = 0 load result into R, load carry out into Temporary Carry (TC)
  - read IXH from RB (bus DA), read constant 00f from BR (bus DB), load Result into AL

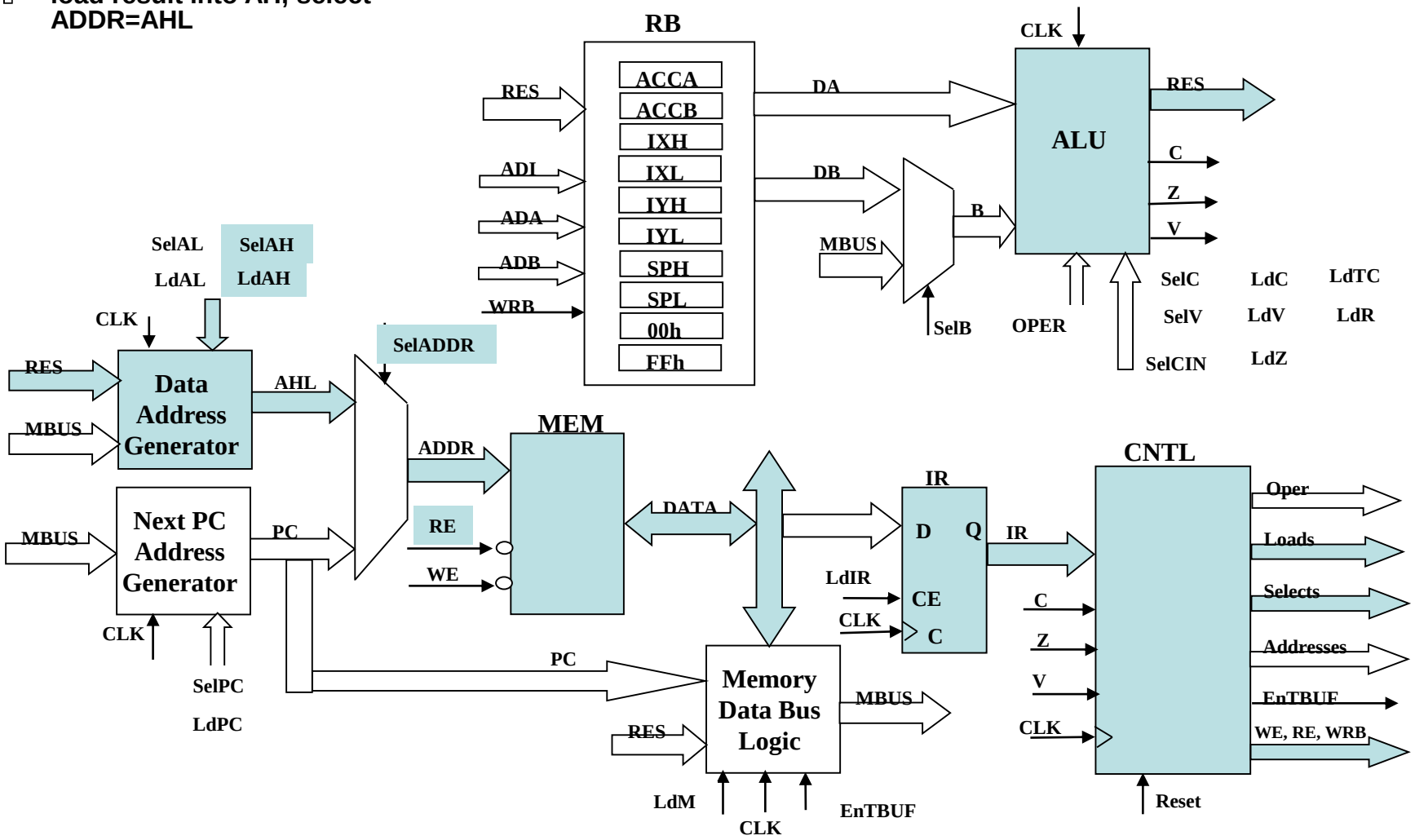


- **E Clock #2**
  - load offset into M register, read IXL from RB, disable address
  - add IXL to offset in M, CIN = 0 load result into R, load carry out into Temporary Carry (TC)
  - read IXH from RB (bus DA), read constant 00f from BR (bus DB), load Result into AL

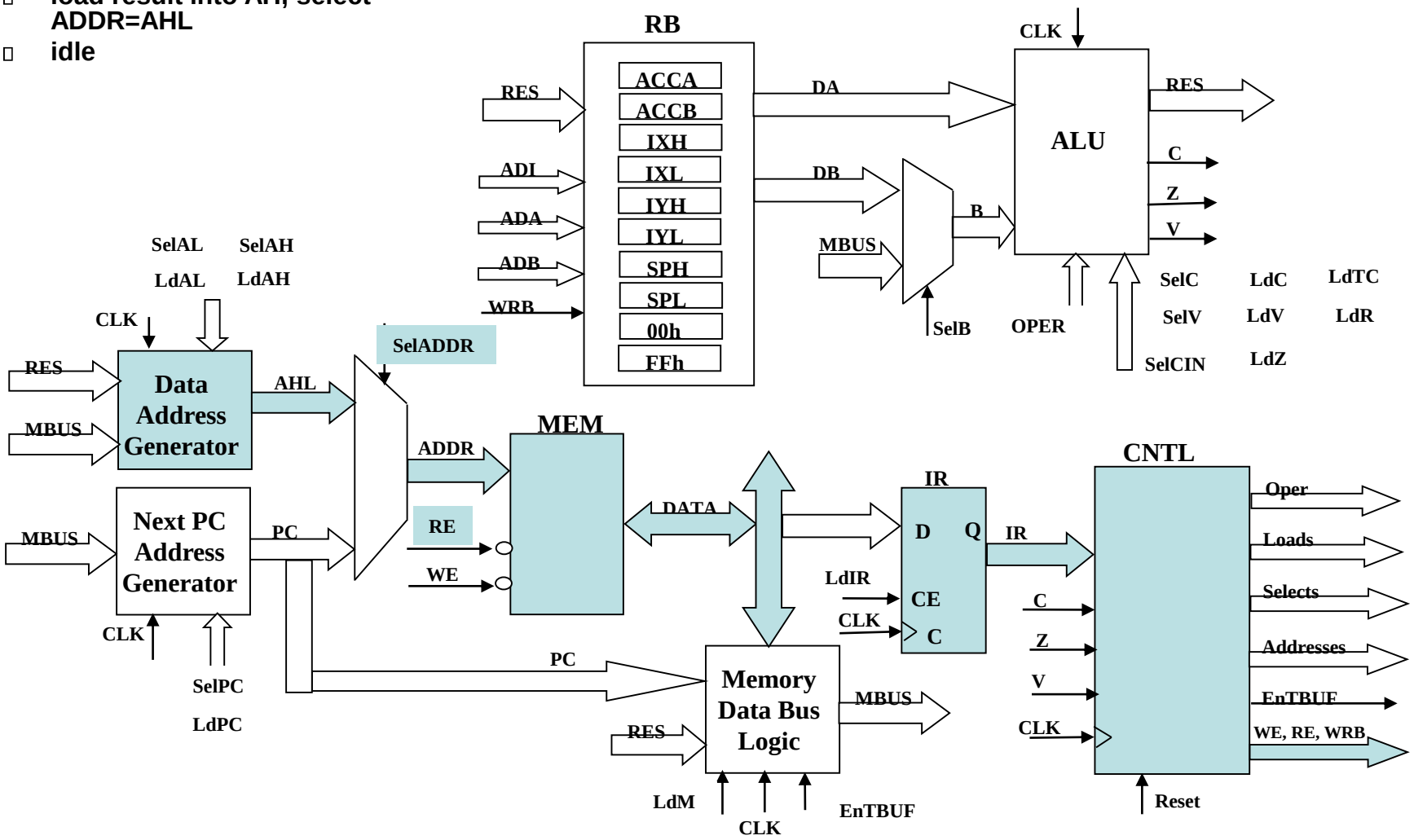


- add IXH to constant 00h, CIN = TC, load result into R

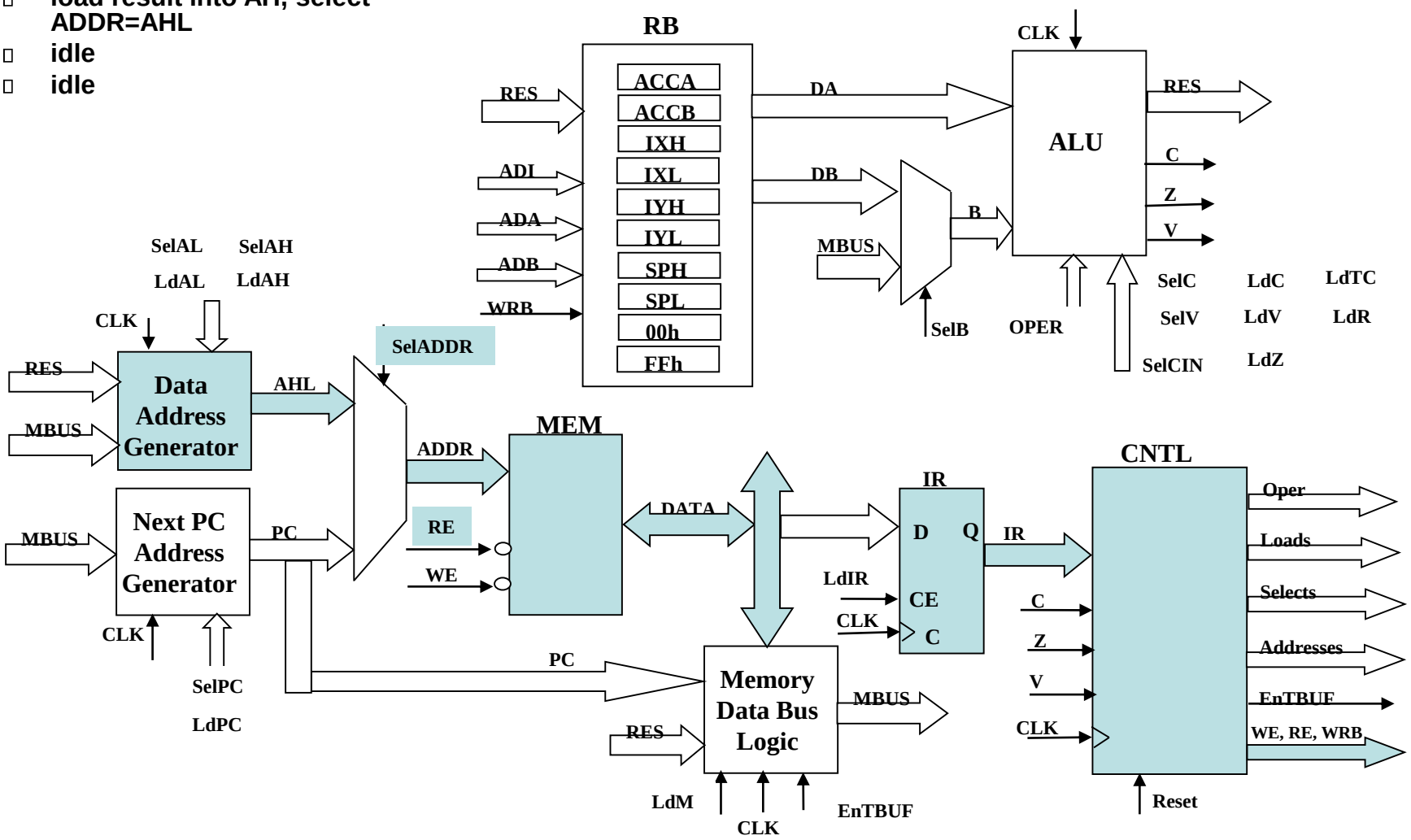
- **E Clock #3**
  - **load result into AH, select ADDR=AHL**



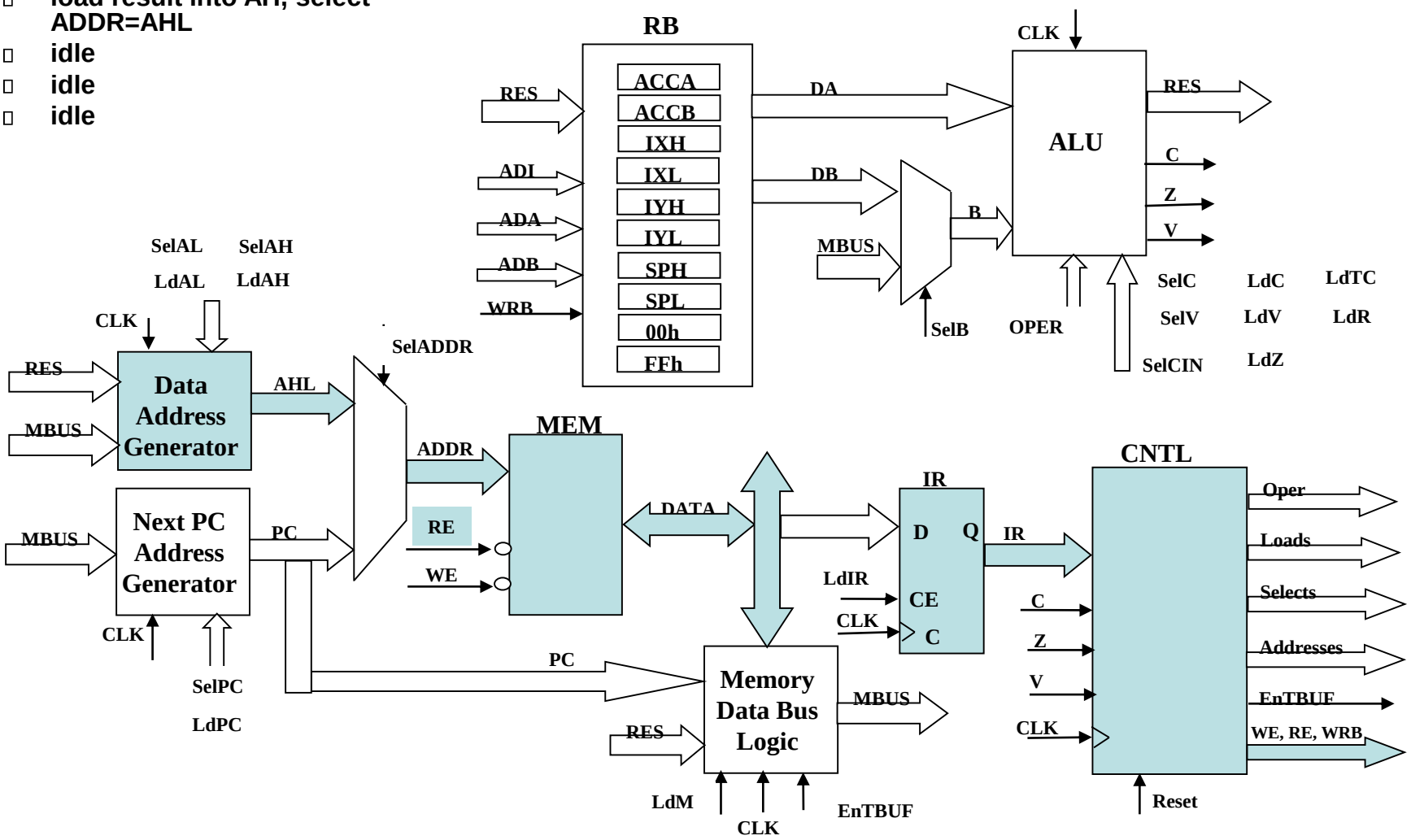
- **E Clock #3**
  - load result into AH, select ADDR=AHL
  - idle



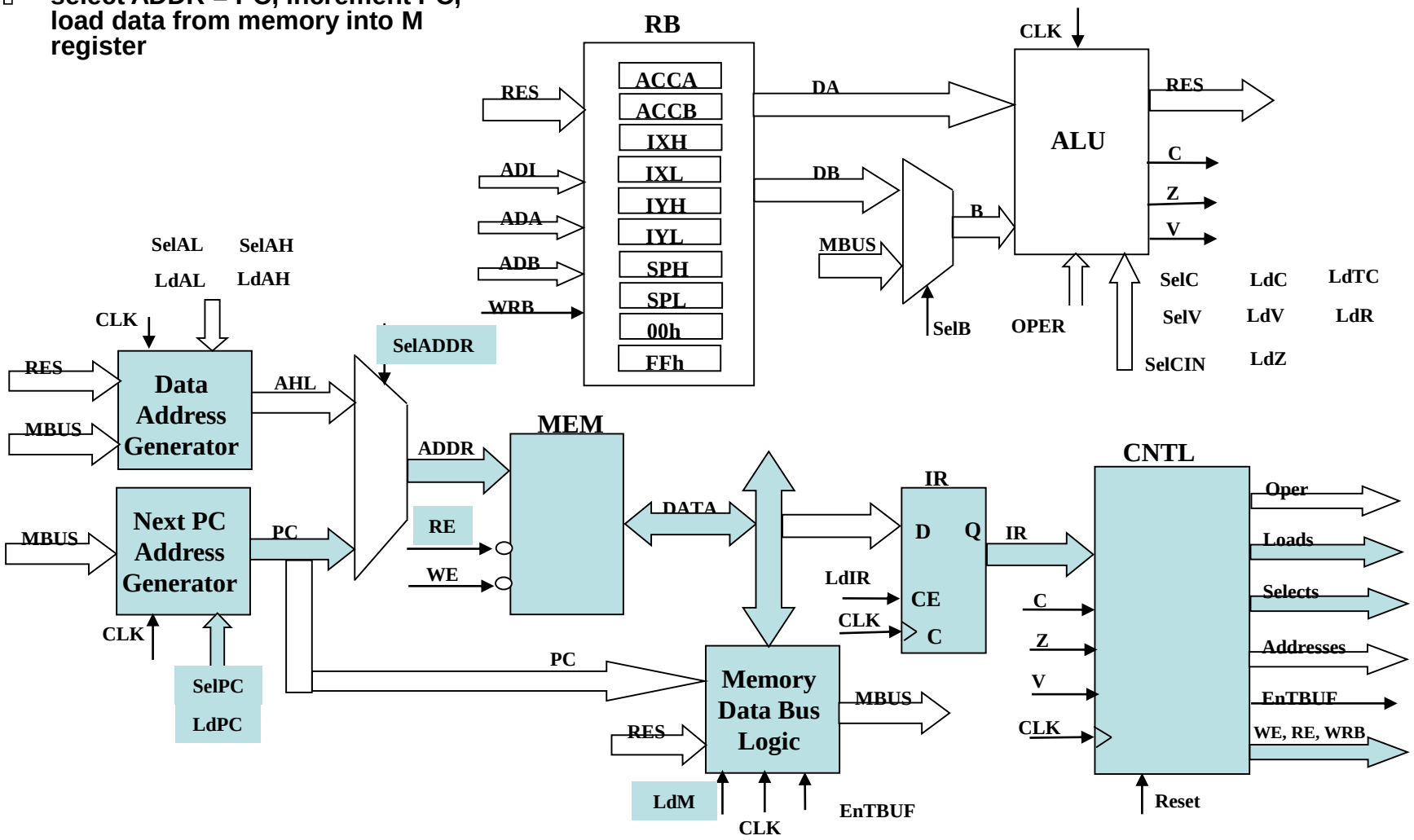
- **E Clock #3**
  - load result into AH, select ADDR=AHL
  - idle
  - idle



- **E Clock #3**
  - load result into AH, select ADDR=AHL
  - idle
  - idle
  - idle

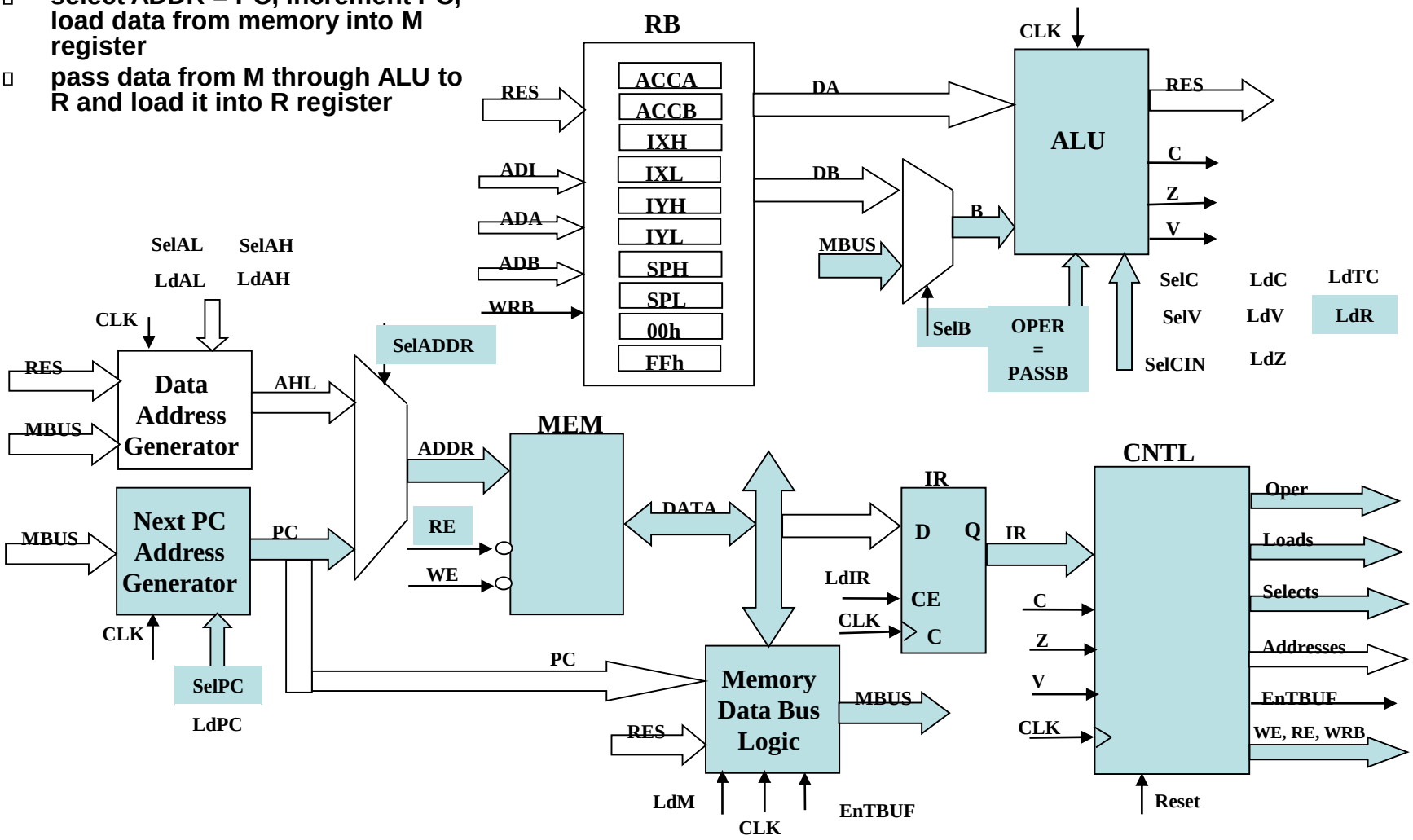


- **E Clock #4**
  - **select ADDR = PC, increment PC, load data from memory into M register**

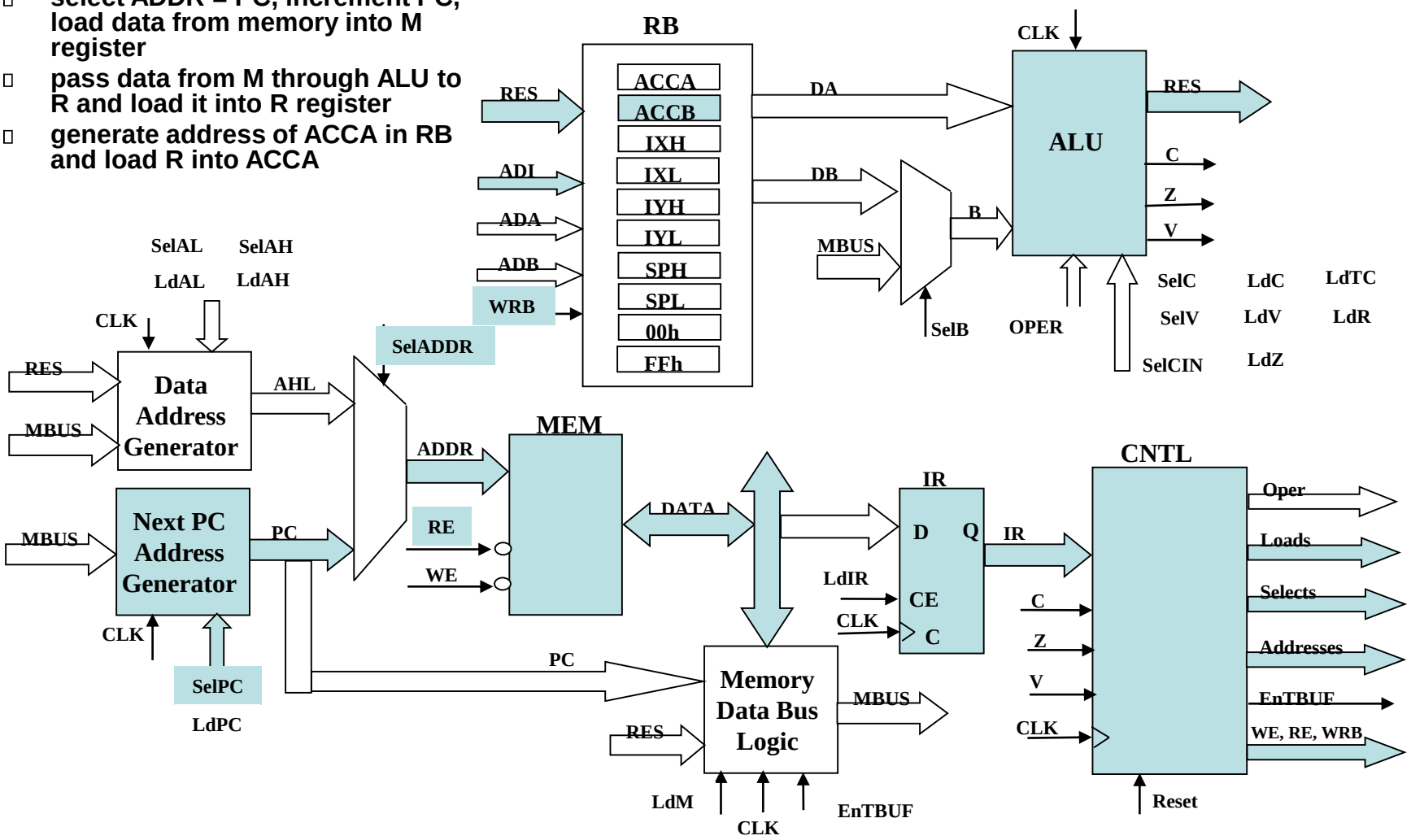


HC11: Instrukce LDAB, adresovací režim IND, X

- **E Clock #4**
  - **select ADDR = PC, increment PC, load data from memory into M register**
  - **pass data from M through ALU to R and load it into R register**



- **E Clock #4**
  - select ADDR = PC, increment PC, load data from memory into M register
  - pass data from M through ALU to R and load it into R register
  - generate address of ACCA in RB and load R into ACCA



HC11: Instrukce LDAB, adresovací režim IND, X

- **E Clock #4**
  - select ADDR = PC, increment PC, load data from memory into M register
  - pass data from M through ALU to R and load it into R register
  - generate address of ACCA in RB and load R into ACCA
  - select V=VOUT, load V and Z flags, (N flag not implemented here)