Pr-6 3

9. Strojově orientované jazyky

Programování – 6
Verze 1.1

5

David Martinek, 2017–2021

Gymnázium Brno, Vídeňská

Pr-6 3

Strojově orientované jazyky

- Schémata fungování počítače
  - Von Neumannovo schéma
- Instrukce, instrukční cyklus
- Instrukční sady CISC, RISC
- Strojově orientované jazyky
- Způsoby adresování paměti

Pr-6 3

Cíl této kapitoly

Zjistit, jak funguje programovatelný počítač na úrovni procesoru.

Pr-6 3

Schémata fungování počítače

- Von Neumannovo schéma
  - John von Neumann, 1945
  - Program i data leží ve stejné paměti.
  - V jednu chvíli lze číst/zapisovat vždy jen instrukci nebo data → sekvenční zpracování.
  - Jednodušší, proto se rozšířila v počátcích počítačové doby.
  - Pomalejší než Harwardská architektura.

Obrázek viz wikipedia.org

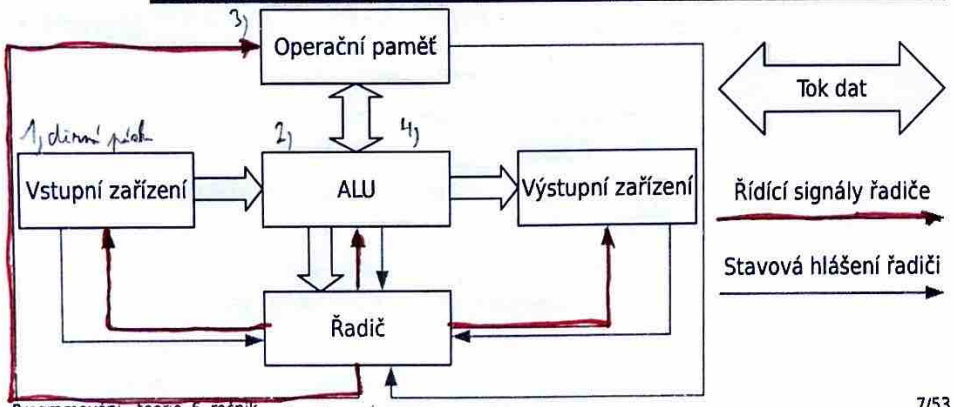
Schémata fungování počítače

- **Harvardská architektura**
  - Počítač Harvard Mark I, 1944 (elektromechanický).
  - **Oddělené paměti programu a dat**
    - Mohou mít různé parametry (rychlost, technologii, adresování)
  - Instrukce i data lze zpracovávat paralelně.
  - Využívá se např. u signálových procesorů, jednoúčelových počítačů.
    - Program může být v paměti typu ROM.

Schémata fungování počítače

- Moderní procesory
  - Spojují rysy obou schémat.
  - Vevnitř využívají rysy Harvardské architektury a pracují paralelně (na více úrovních).
  - Navenek se chovají podle von Neumannova schématu, protože využívají stejnou sběrnici pro čtení dat i programu z hlavní paměti.

Von Neumannovo schéma



Von Neumannovo schéma

- **ALU – aritmeticko-logická jednotka**
  - Provádí veškeré aritmetické a logické operace.
  - Obsahuje sčítačky, násobičky a komparátory.
- **Řadič**
  - Řídící jednotka
  - Řídí činnost všech modulů pomocí řídících signálů.
  - Moduly posílají své reakce na řídící signály zpět pomocí stavových hlášení.

Pr-6 3 Von Neumannovo schéma

- Operační paměť
  - Obsahuje zpracovávaná data, výsledky i samotný program.
  - *Mezi daty a programem není principiální rozdíl!*
- Vstupní zařízení
  - Zařízení pro vstup dat a programů
- Výstupní zařízení
  - Zařízení pro výstup výsledků programu

Pr-6 3 Von Neumannovo schéma

- Procesor
  - Řadič + ALU
- CPU – central processor unit
  - Procesor + operační paměť (registry, cache, ...) + další koprocesory (FPU – floating point unit, vektorové jednotky SIMD – single instruction multiple data, ...)

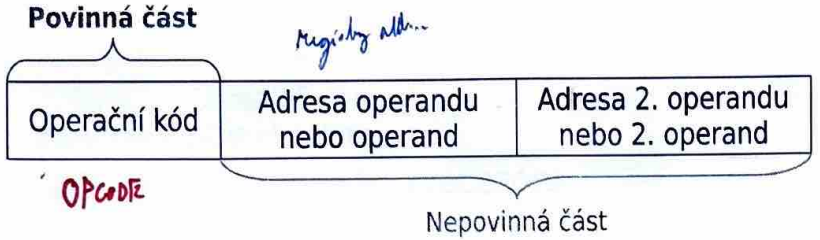
Pr-6 3 Von Neumannovo schéma

- Princip činnosti počítače podle von Neumanna
  1. Zavedení programu
     - Ze vstupního zařízení pomocí ALU do operační paměti
  2. Zavedení dat
     - Stejným způsobem jako program
  3. Výpočet
     - Řadič + ALU vykonávají instrukce programu čtené z operační paměti.
     - Mezivýsledky se ukládají do operační paměti.
  4. Po skončení výpočtu se výsledky pošlou přes ALU na výstupní zařízení.

Pr-6 3 Instrukce

- Předpis pro provedení nějaké elementární činnosti
- Činnost je „naprogramována“ vytvořením odpovídajících logických obvodů při návrhu procesoru.
- V moderních procesorech je vnitřně rozdělena na menší části – mikroinstrukce, které mohou být zpracovávány paralelně.

Struktura instrukce

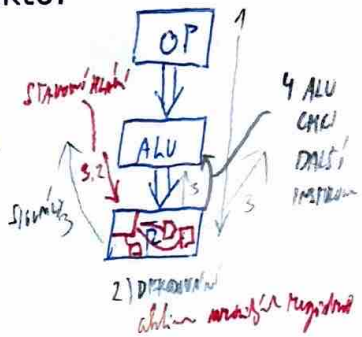


MOV EAX, DS:[EBX]

Von Neumannův instrukční cyklus

- Program řídí řadič mikroprocesoru takto:

1) Převzetí instrukce z operační paměti
2) Dekódování instrukce
   - Řadič obsahuje modul (obvod) *dekodér instrukcí.*
3) Provedení operace
   - Řadič posílá řídící signály ALU a jiným modulům.
4) Příprava k převzetí další instrukce



Instrukční sada

- Množina všech instrukcí daného procesoru
- Různé typy procesorů mají různé instrukční sady.
- Některé skupiny procesorů mají podobný návrh instrukční sady (architekturu).
  - CISC – Complete Instruction Set Computer
  - RISC – Reduced Instruction Set Computer

CISC

- **C**omplete **I**nstruction **S**et **C**omputer
- Instrukce řeší velmi široký okruh operací.
  - Obvykle vnitřně realizováno pomocí tzv. mikroinstrukcí.
- Procesor umí až *stovky různých instrukcí*.
- Instrukce jsou vykonávány *různě dlouhou dobu*.
  - Potřebují různý počet taktů procesoru.
- Instrukce jsou *zpracovávány sériově*.
  - V jednom okamžiku se vykonává vždy jen jedna instrukce.

Pr-6 3

CISC

- Výhodné pro programátory (lidi) (?)
  - Instrukce tvoří „knihovnu“ různých operací.
  - Ukázalo se, že reálně se využívá malá část instrukcí ⇒ impuls pro vznik RISC architektur.
- Vede ke složitému návrhu procesoru.
- Dnešní stav
  - CISC: vnější slupka procesoru
  - RISC: vnitřní jádro (jádra) procesoru

Pr-6 3

RISC

- **R**educed **I**nstruction **S**et **C**omputer
  - Architektura původně navržena pro superpočítače.
  - Vznikla jako reakce na pomalé a neefektivní CISC procesory.
  - Cca 80. léta 20. století
- Malý počet jednoduchých instrukcí
  - minimum základních operací

Pr-6 3

RISC – optimalizace

- Pipelining
  - Zřetězení či překrývání instrukcí (jejich částí – mikroinstrukcí)
  - Částečně paralelní vykonávání instrukcí
  - Obecný princip (ne jen v IT)
    - Rozdíl v efektivitě jako mezi manufakturou (sériové zpracování instrukcí) a pásovou výrobou (pipelining)
- V každém taktu procesoru dokončena jedna instrukce!

Pr-6 3

RISC

- Nutno používat kvalitní překladač.
  - Ruční programování pomocí JSI je obtížné.
- Rychlejší běh programů
- Efektivnější využití zdrojů
- Méně součástek pro instrukce ⇒ více místa pro další paměť a paralelní, duplikované moduly

assembler = jazyk strojových instrukcí
textový ekvivalent = jazyk symbolických instrukcí

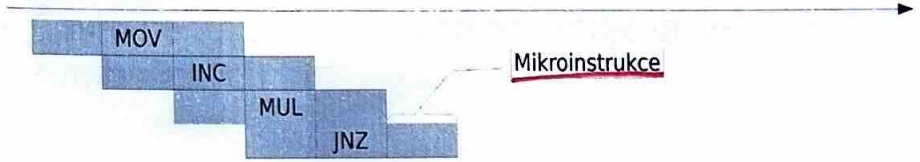
Pr-6 3

CISC vs. RISC

znát

- CISC + sériové zpracování

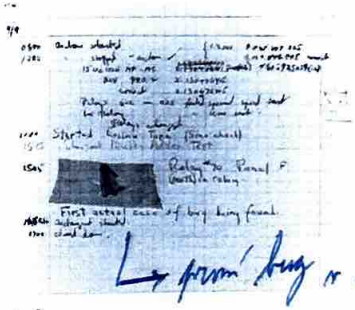
- RISC + pipelining (urychlení)



Pr-6 3

Strojově orientované jazyky

- První počítače (Eniac, Maniac) se programovaly mechanicky – pomocí elektrických propojek.
  - Pracné, náchylné k chybám a bugům.
- U další generace počítačů se textový zápis instrukcí pro počítač převáděl na děrné štítky a pásky.
  - Zvětšení komfortu a spolehlivosti.
- Dnes principiálně podobnou činnost dělají překladače programovacích jazyků.



Pr-6 3

Strojový kód (SK)

- Jazyk stroje = binární kód (přímé instrukce)
- Skládá se z dat a instrukcí uložených binárně.
- Nejnižší programovací jazyk
- Není určen pro lidi – je generován z vyšších programovacích jazyků pomocí překladače a assembleru.
- Je uložen v operační paměti.
- Je vykonáván procesorem (ALU + řadič).

Pr-6 3

Jazyk symbolických instrukcí (JSI)

- Anglický termín: *assembly language*
- Alternativní český název: jazyk symbolických adres
- Textová reprezentace strojového kódu
- Určen pro programátory (je srozumitelný).
- Nejnižší programovací jazyk použitelný člověkem
- Tvořen souborem instrukcí a pravidly jejich zápisu.
- Existují různé varianty pro různé architektury mikroprocesorů (Intel, Motorola, ...).

Pr-6 3

Assembler (ASM)

PŘEKLADAČ — NIKOLIV JAZYK

- Program pro překlad jazyka symbolických instrukcí do strojového kódu.
- Často se takto nesprávně označuje jazyk symbolických instrukcí.

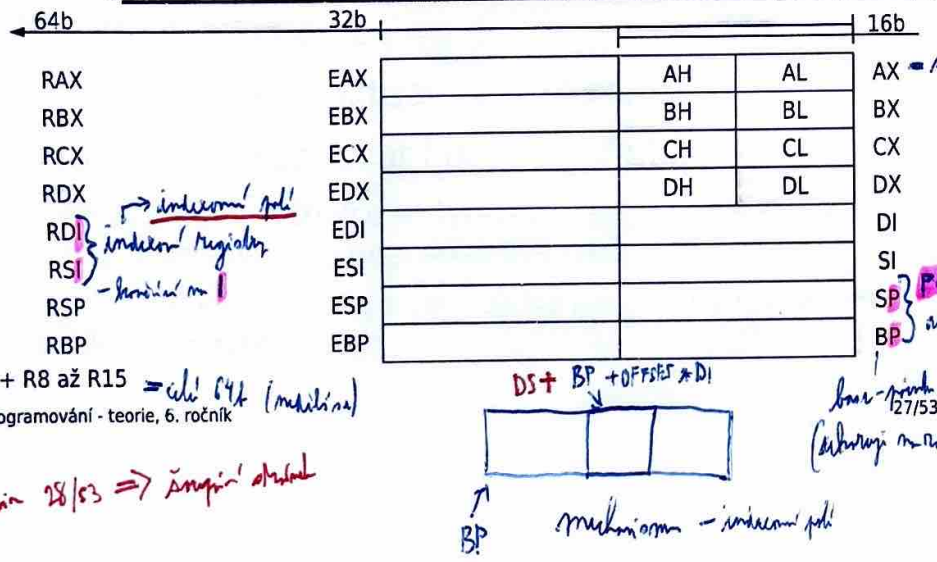
Pr-6 3

Registry procesoru

- Nejnižší a nejrychlejší úroveň paměti počítače
  - Z pohledu JSI jsou to nejrychlejší *proměnné*.
  - Klíčový způsob práce s daty v jádru procesoru.
- Obsahují operandy instrukcí
  - Číselné hodnoty
  - Adresy paměťových buněk (s RAM se nedá pracovat přímo)
- Po vykonání instrukce se do nich zapisují výsledky.
- Existují různé typy registrů
  - Univerzální, specializované, matematické, vektorové, ...

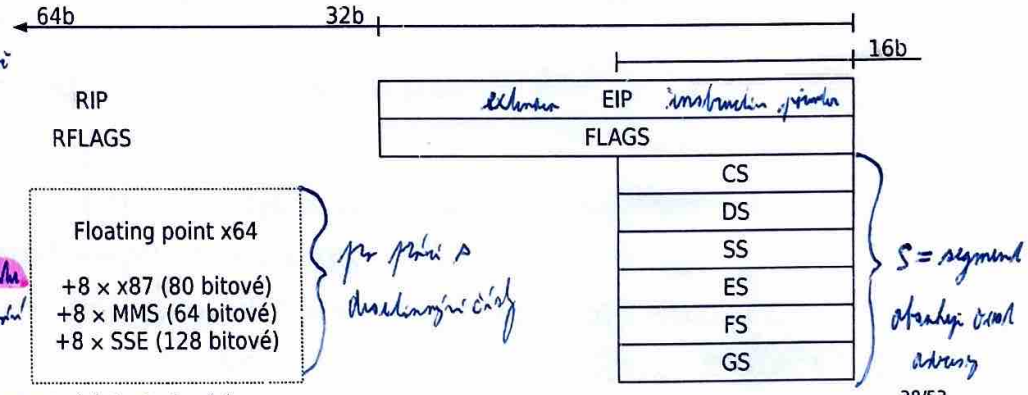
Pr-6 3

Registry Intel 486 (a výš)



Pr-6 3

Registry Intel 486 (a výš)



Pr-6 3 Univerzální registry

- 8b: AH, AL, BH, BL, CH, CL, DH, DL
- 16b: AX, BX, CX, DX, SI, DI, BP, SP
- 32b: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP

} univerzální

- AX, EAX
  - Akumulátor/střadač
  - Nejčastěji používaný registr
  - Často implicitní registr pro ukládání výsledků

Pr-6 3 Segmentové registry

- 16b: DS (data segment), CS (code segment), SS (stack segment), ES (extra), FS, GS
- Pro adresování segmentů paměti
- Uchovávají adresu začátku segmentu.

JMP LABEL; Skok na adresu LABEL ⇒ změní hodnotu registru EIP/RIP

Pr-6 3 Registr *čítač instrukcí*

- IP, EIP (instruction pointer)
- Obsahuje adresu aktuální instukce.
- Nelze do něj kopírovat hodnotu přímo.
  - Po vykonání instrukce se typicky posune na další instrukci (viz 4. krok Von Neumannova instrukčního cyklu).
  - Skokové instrukce jej nastavují na adresu instrukce dané cílovým návěštím (viz dále).

Pr-6 3 Příznakový registr FLAGS

- Registr příznaků různých stavů
- Slouží pro testování a podmíněné skoky.
- ZF (zero flag)
  - Nastaven, když je výsledek předchozí operace 0.
- SF (signum – znaménko),
  - Nastaven, když je výsledkem operace záporné číslo.
- CF (carry flag – přenos), OF (overflow – přetečení), ...

↳ při součtu 0 a 1
↳ přičtení ⇒ nula 1

Pr-6 3 Další registry

- BP, BX, EBP, EBX
  - Bázové registry
  - Pro adresování
- DI, SI
  - Indexové registry
  - Pro snazší adresování polí

Pr-6 3 Demonstrační JSI pro SASM

- SASM
  - **S**koro**AS**se**M**bler
  - Fiktivní assembler pro fiktivní 16 bitový počítač SCPU
  - Pro představu, jak se programuje na nejnižší úrovni
- Demonstrační SkoroCPU
  - 16 bitový procesor
  - Má všechny dříve zmíněné registry, kromě 32b verzí.

Pr-6 3 SASM – instrukce pro vstup a výstup

Instrukce	Popis	Nastavuje FLAGS
IN	Přečte jeden znak (ASCII kód) ze vstupu do registru AL.	ZF, SF
OUT	Zapíše jeden znak z registru AL na výstup.	

Pr-6 3 SASM – skokové instrukce

Instrukce	Popis
JMP n	Nepodmíněný skok na návěští n

- Návěští
  - identifikátor:
  - označení řádku zdrojového textu
  - SASM za něj dosadí adresu následující instrukce

```
// nekonečný cyklus
start:
IN
OUT
JMP start
```

Pr-6 3

SASM – skokové instrukce

Instrukce	Popis
JZ n	Skoč na návěští n, když je nastaven příznak ZF.
JNZ n	Skoč na návěští n, když není nastaven příznak ZF.
JS n	Skoč na návěští n, když je nastaven příznak SF.
JNS n	Skoč na návěští n, když není nastaven příznak SF.

- Testují příznaky ve FLAGS.
  - ZF – zero flag – Výsledkem předchozí operace je 0.
  - SF – signum flag – Výsledkem předchozí operace je záporné číslo.

Pr-6 3

SASM – skokové instrukce

Instrukce	Popis
CALL n	• Vyvolá podprogram začínající na návěští n. • Uloží na zásobník návratovou adresu.
RET	• Ukončí podprogram. • Skočí na návratovou adresu uloženou na zásobníku.

Pr-6 3

SASM – logické instrukce

Instrukce	Popis	Nastavuje FLAGS
CMP x, y	Porovná x a y, nastaví příznaky.	ZF, LF, GF
JL n	Skoč, pokud je menší.	
JLE n	Skoč, pokud je menší či rovno.	
JG n	Skoč, pokud je větší.	
JGE n	Skoč, pokud je větší či rovno.	

- Skoky se volají po CMP a testují výsledek porovnání podle příznaků ve FLAGS.
- JZ n slouží zároveň jako skoč, pokud je rovno.

Pr-6 3

SASM – aritmetické instrukce

Instrukce	Popis	Nastavuje příznaky ve FLAGS
INC x	x ← x + 1	ZF, SF
DEC x	x ← x - 1	ZF, SF
ADD x, y	x ← x + y	ZF, SF
SUB x, y	x ← x - y	ZF, SF
MPL x, y	x ← x * y	ZF, SF
DIV x, y	x ← x / y	ZF, SF
MOD x, y	x ← x mod y	ZF, SF

Pr-6 3

SASM – přesun hodnot

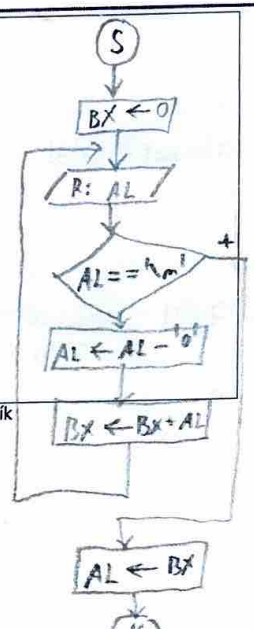
Instrukce	Popis	Nastavuje příznaky ve FLAGS
MOV x, y	x ← y	ZF, SF
SWAP x, y	x ↔ y (ne konstanty)	ZF, SF (podle x)

- Parametry instrukcí
  - x, y – registry, adresy paměťových buněk
  - y – Může být konstanta (ne u instrukce SWAP).
  - x – Musí to být větší nebo stejný prostor jako y.
    - ADD AL, AX nejde; ADD AX, AL ano

Pr-6 3

SASM – ukázkový příklad

```
nacticislo:
  MOV BX,0
cyklus:
  IN
  CMP AL,'\n'
  JZ end
  SUB AL,'0'
  ADD BX,AL
  JMP cyklus
end:
  MOV AX,BX
  RET
```



- Co tento kód dělá?
- Jak?
- Kde bude uložen výsledek?

Pr-6 3

Způsoby adresování paměti

- INSTRUKCE operand1, operand2
- Operand
  - Přímý operand (konstanta)
  - Registr
  - Adresa paměťové buňky
- Existují různé způsoby výběru paměťových míst.
- Tyto způsoby jsou zakódovány v kódu instrukce.

Pr-6 3

Způsoby adresování paměti

1. Přímý operand
   - Adresování 0. řádu
   - Bez odkazu – operand umístěn přímo v instrukci.
2. Přímé adresování
   - Adresování 1. řádu
   - Operand je určen pomocí absolutní adresy paměťového místa.

Bázový registr = Base Register

{Pr-6 3} Způsoby adresování paměti

3. Registr
   - Operandem je registr
   - Adresování 2. řádu
4. Relativní adresování
   - Adresování s bázovým registrem
   - Adresa = bázový registr + relativní posun
   - Program je pak beze změn přemístitelný v paměti – mění se jen obsah bázových registrů.

{Pr-6 3} Způsoby adresování paměti

5. Stránkové adresování
   - Relativní adresování se segmentovým registrem
   - Paměť je rozdělena na stránky – segmenty.
6. Indexové adresování
   - Adresování s indexovým registrem
   - Pro indexování polí – při průchodu polem se zvyšuje indexový registr.

MOV AX, [BASE + SI];

JMP LABEL;

{Pr-6 3} Způsoby adresování paměti

7. Nepřímé adresování
   - Ve spojení se všemi ostatními metodami adresování
   - Paměťové místo určené předchozími metodami obsahuje nikoliv operand, ale adresu.

{Pr-6 3} Způsoby adresování paměti

$$\begin{pmatrix} CS: \\ DS: \\ SS: \\ ES: \\ FS: \\ GS: \end{pmatrix} \left[\begin{pmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{pmatrix} + \left[\begin{pmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{pmatrix} * \begin{pmatrix} 1 \\ 2 \\ 4 \\ 8 \end{pmatrix} \right] + [posuv] \right]$$

{ Pr-6 3 } Otázky

- Nakresli Von Neumannovo schéma počítače. Popiš a vysvětli funkci všech bloků a signálů.
- Jaké jsou 2 nejdůležitější rozdíly mezi Von Neumannovou a Harvardskou architekturou?
- Popiš a vysvětli princip činnosti počítače podle Von Neumanna. V jakém pořadí se používají jednotlivé bloky takového počítače?

{ Pr-6 3 } Otázky

- Vysvětli pojem instrukce a popiš Von Neumannův instrukční cyklus.
- Vysvětli pojem instrukční sada. Charakterizuj CISC a RISC.
- Jak a čím je vytvářen elementární program (strojový kód) pro procesor počítače. Vysvětli tři související pojmy.

{ Pr-6 3 } Otázky

- Charakterizuj skupiny registrů procesoru (Intel 486+).
- Vysvětli pojmy relativní adresování, stránkové adresování a indexové adresování. Které registry se jich účastní?
- Popiš rozdíly mezi přímým, relativním a nepřímým adresováním.

{ Pr-6 3 } Otázky

- Napiš pomocí JSI program, který vytiskne dvě hodnoty x, y v pořadí od nejmenší po největší. Na začátku budou hodnoty x, y uloženy v registrech AX, BX.
- Napiš pomocí JSI program, který na výstup vytiskne n krát znak c. Znak c zadá uživatel na vstupu a hodnota n bude na začátku uložena v registru AX.

Pr-6 3

Otázky

- Napiš pomocí JSI program, který vypočítá součet přirozených čísel mezi hodnotami x a y (x + (x+1)+ (x+2)+...+y). Na začátku bude hodnota x uložena v registru BX a y v CX. Výsledek nechť je v AX.
- Napiš pomocí JSI program, který vytiskne největší ze dvou hodnot a, b. Na začátku budou hodnoty a, b uloženy v registrech AX, BX.

vytiskne prvních N členů posloupnosti. N je v (AX) – registr AX

```
n = AX
a = 'A'
while (n>0) {
    print(a);
    a++;
    n--;
}
```

KÓD

```
    MOV BX, AX
    MOV AL, 'A'
while:
    CMP BX, 0
    JLE end while
    OUT
    INC AL
    DEC BX
    JMP while
end while:
    RET
```

jsou stejné

KÓD

```
if (a+b > a*b)      // AL ... a
    print('A');     // BL ... b
else
    print('N');
```

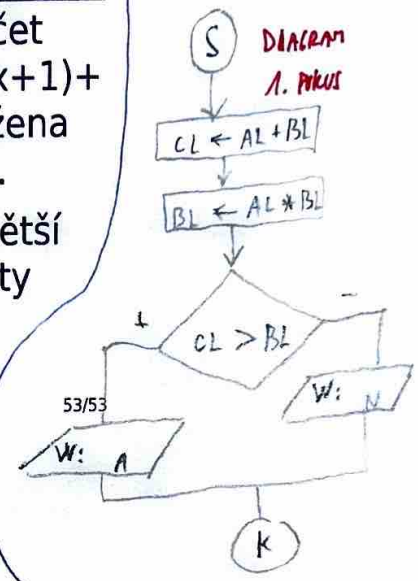


DIAGRAM 1. příkaz

```
    MOV CL, AL
    ADD CL, BL
    MPL BL, AL
    CMP CL, BL
    JLE else
    MOV AL, 'A'
    JMP endif
else:
    MOV AL, 'N'
endif:
    OUT
    REF
```

je menší nebo stejné (pak >) když to neplatí, tak to platilo

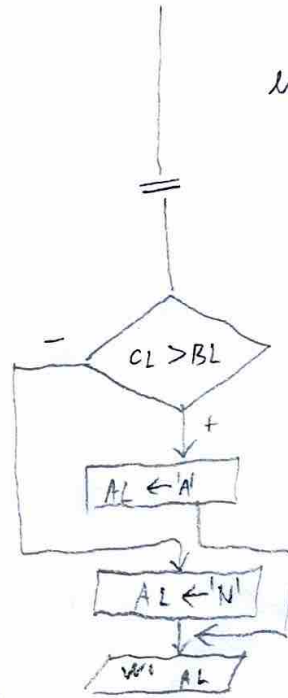


DIAGRAM ODPOVÍDAJÍCÍ INSTRUKCÍM

Pozn: AX

AH	AL

AL je součástí AX