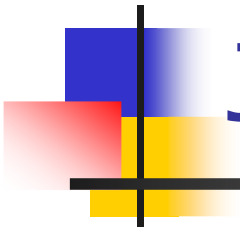


Simulátory aplikačně specifických instrukčních procesorů
Jazyk LISA

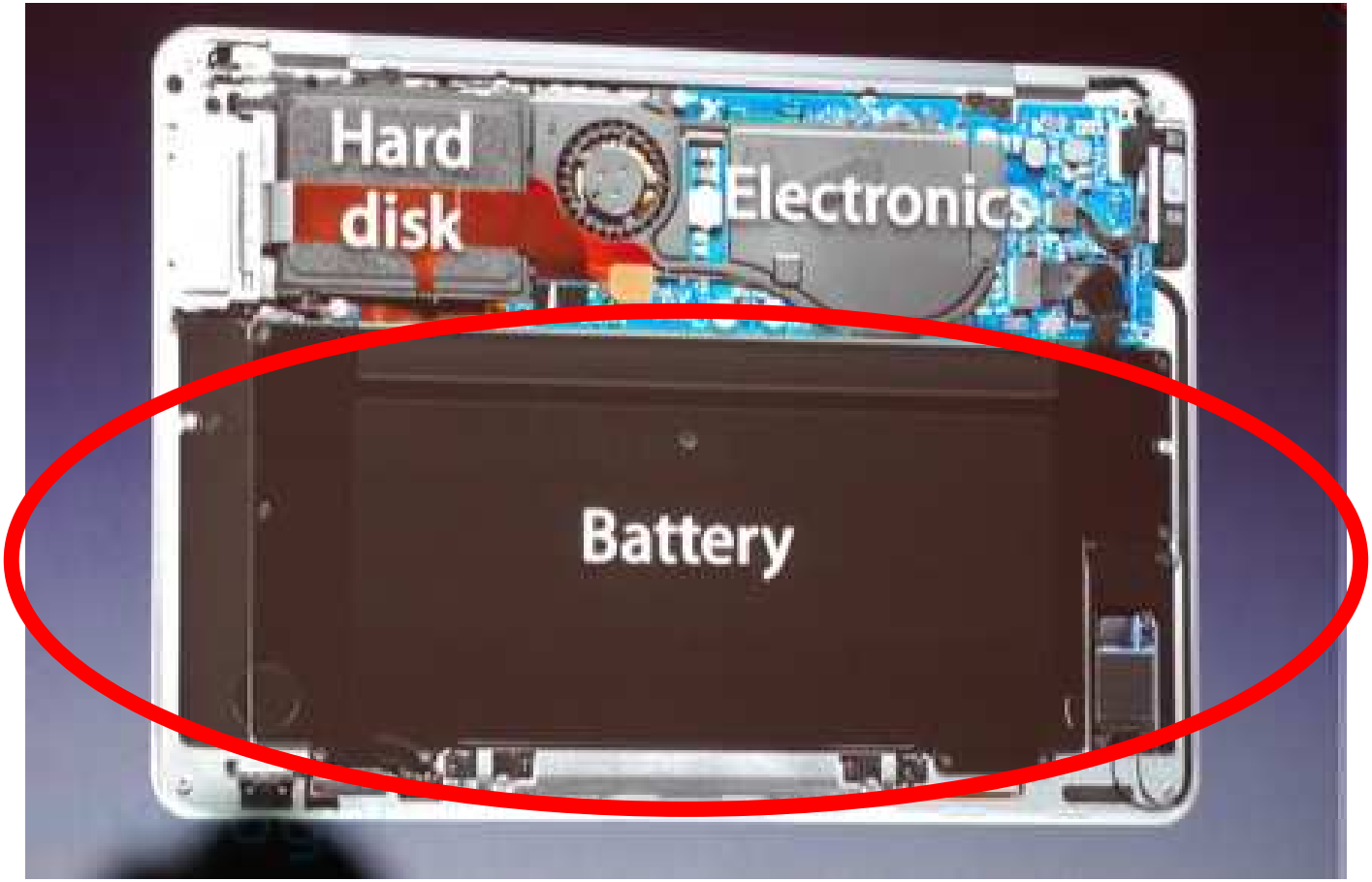
Masařík Karel

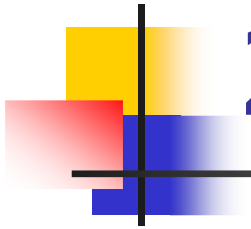
(masarik@fit.vutbr.cz)

1. Úvod

- Vestavěný systém
  - Jednoúčelový systém, ve kterém je řídicí počítač zcela zabudován do zařízení, které ovládá
  - Roste složitost systému
  - Velká rozmanitost typově stejných zařízení
  - Požadavek na nízkou cenu, spotřebu a velikost

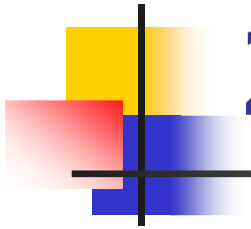
1. Úvod - MacBook Air





2. Návrh vestavěného systému

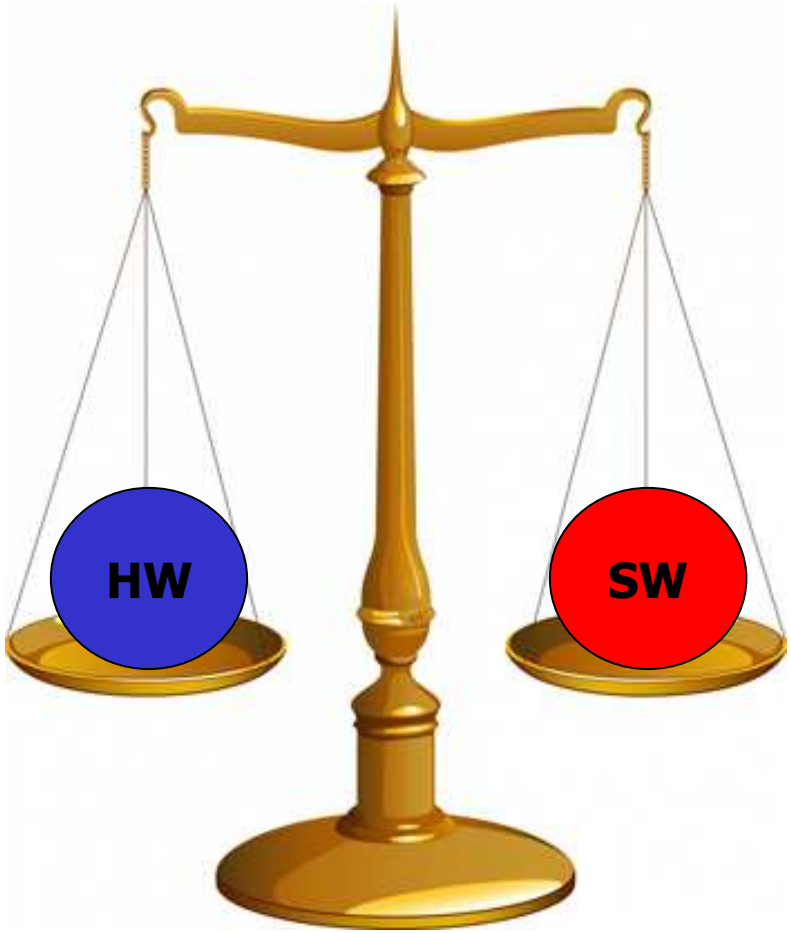
- Úspěch na trhu
  - Inovativní řešení (koukám se co dělá soused na své zahrádce?)
  - Kvalita řešení
    - Cena x výkon
- Cena zařízení ovlivněna
  - Cenou návrhu
  - Cenou výroby
- Kvalita návrhu
  - Závisí na délce návrhu



2. Návrh vestavěného systému

- Vestavěný systém
  - Software+hardware
    - Software pro dosažení multi-fukčnosti zařízení
- Požadujeme
  - Krátký vývojový cyklus (nízká cena)
    - Použijí se dostupná řešení?
  - Nalezení optimálního řešení pro daný problém (dostatečný výkon zařízení)
    - Nutno navrhnout nové vlastní řešení

3. Souběžný návrh hardwaru a softwaru



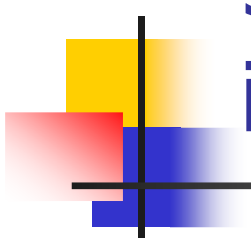
4. Podpora souběžného návrhu hardwaru a softwaru

- Specifikace
  - Model
- Překladač
- **Simulátor**
- Hardware

LISSOM

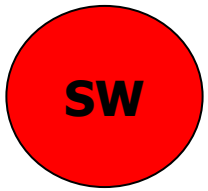
www.fit.vutbr.cz/research/groups/lissom

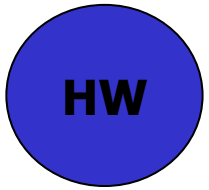
5. Simulace aplikačně specifických instrukčních procesorů



- **ASIP** - Aplikačně specifický instrukční procesor
- **MPSoC** – Víceprocesorový systém na čipu
- **ADL** – Jazyk pro popis architektury
- Proč potřebujeme simulátor při návrhu ASIP?
  - Ověření funkcionality ASIP či MPSoC před vlastním zhotovením
  - Hledání úzkých míst v hardwaru a softwaru
    - Sběr profilovacích dat a optimalizace jak HW tak SW
- Vývoj softwaru pro daný ASIP či MPSoC je mnohem jednodušší na simulátoru než ve skutečném hardwaru
  - Např.: v procesu hledání škodlivého softwaru monitoruje antivirový program chování softwaru

6. Typy simulace ASIP





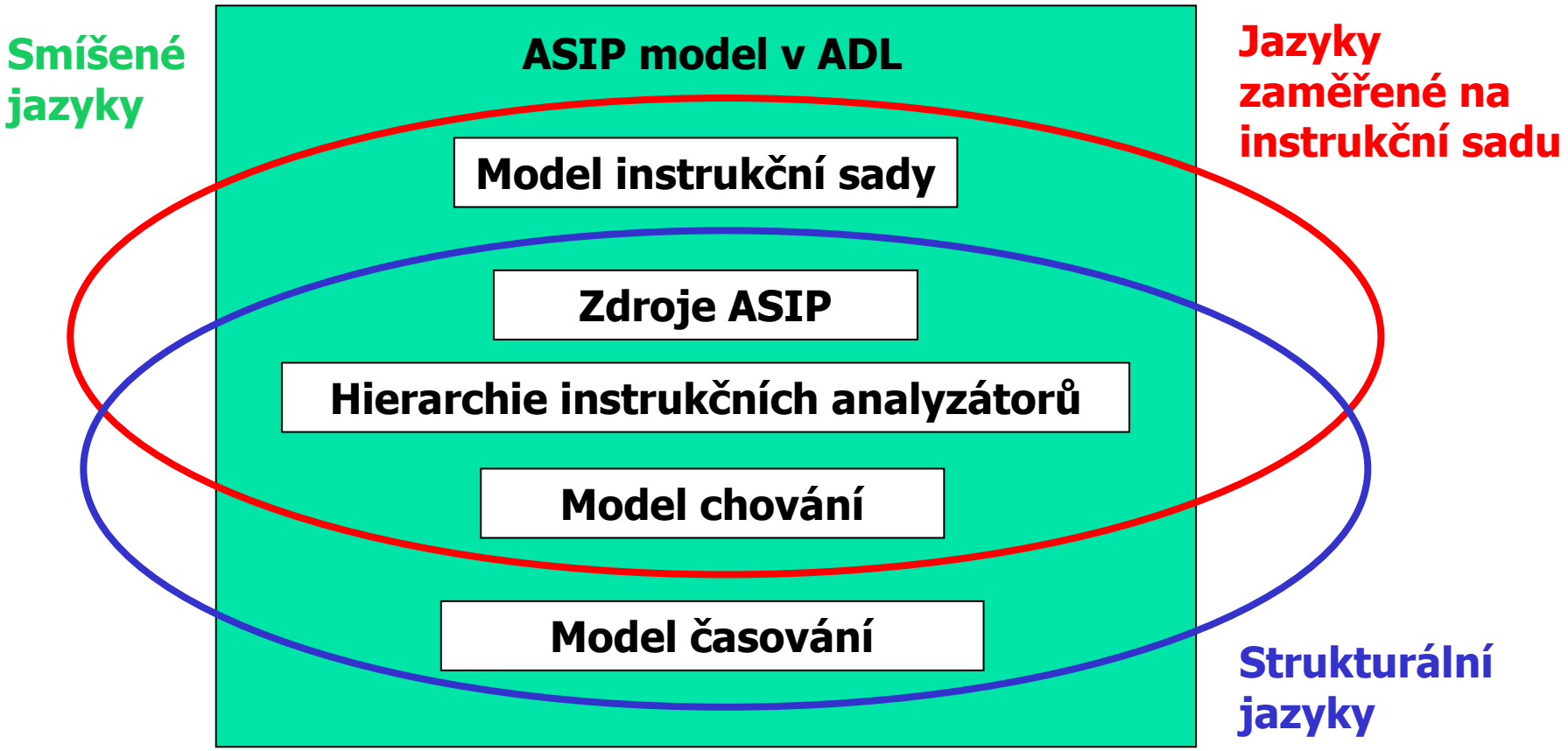
Rychlost

Simulátor založený na instrukcích

Přesnost

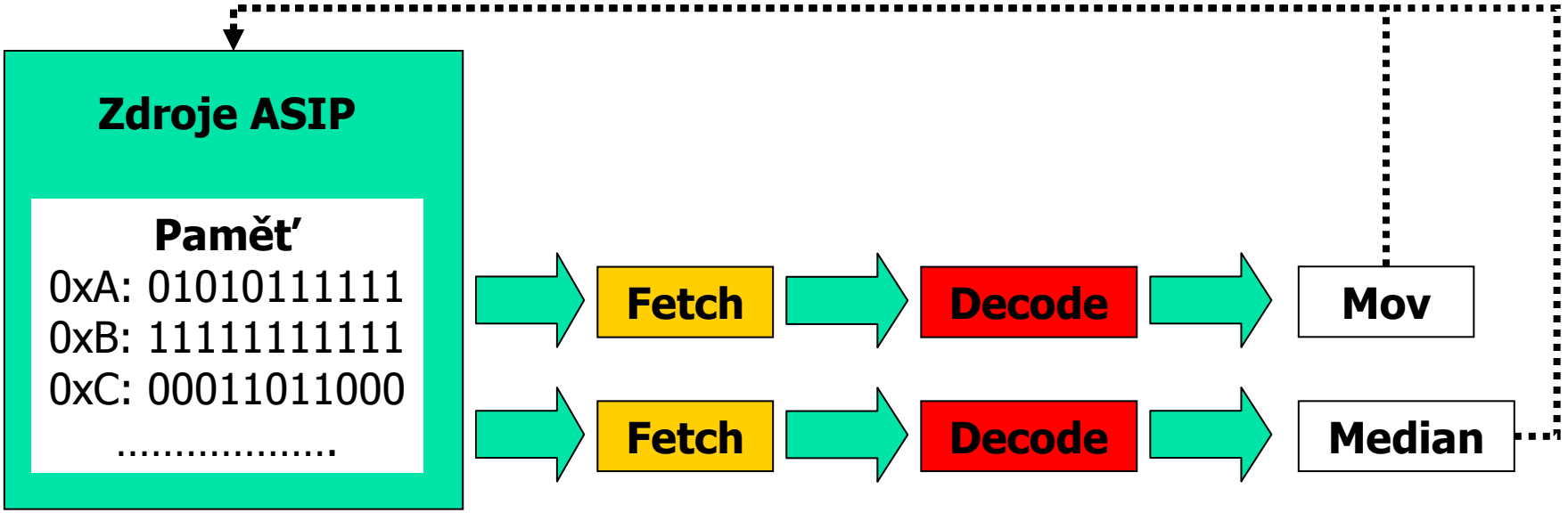
Simulátor založený na cyklech

6. ASIP model v ADL



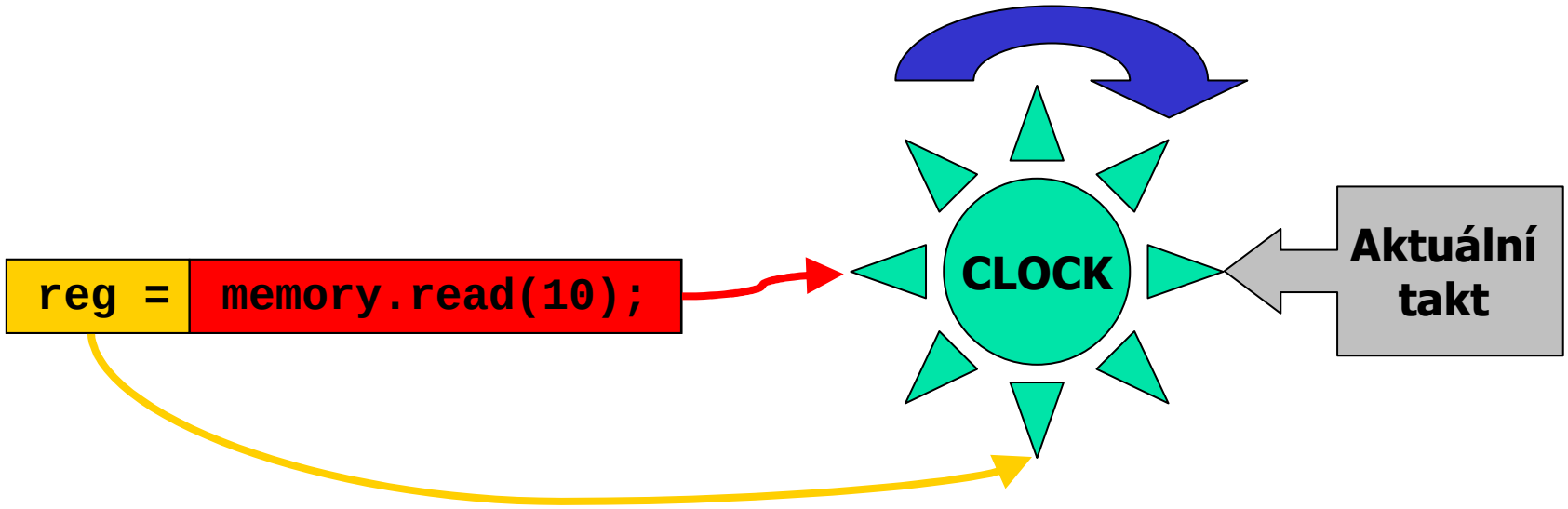
6. Simulátor založený na instrukcích

- Simulace
- Nejmenší krok simulace
  - Provedení jedné instrukce



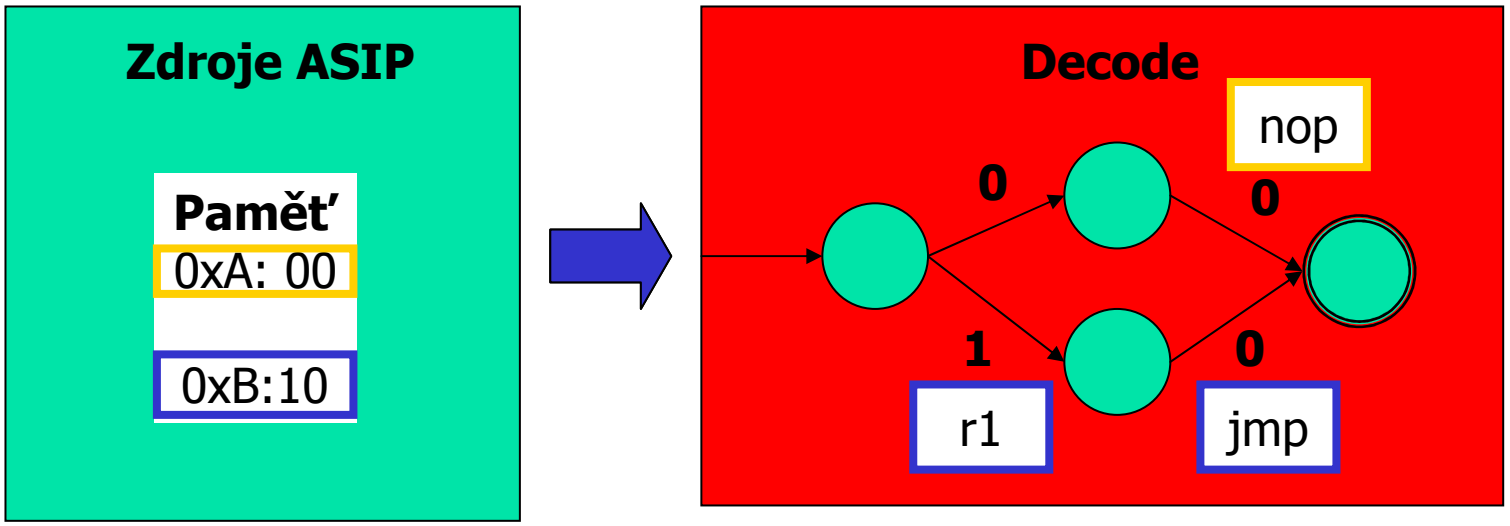
6. Simulátor založený na cyklech

- Simulace
- Nejmenší krok simulace
  - Provedení jednoho hodinového taktu
- Instrukce se provádí několik taktů
  - Přístup ke zdrojům může trvat i několik taktů
    - Např. simulace vyrovnávacích pamětí



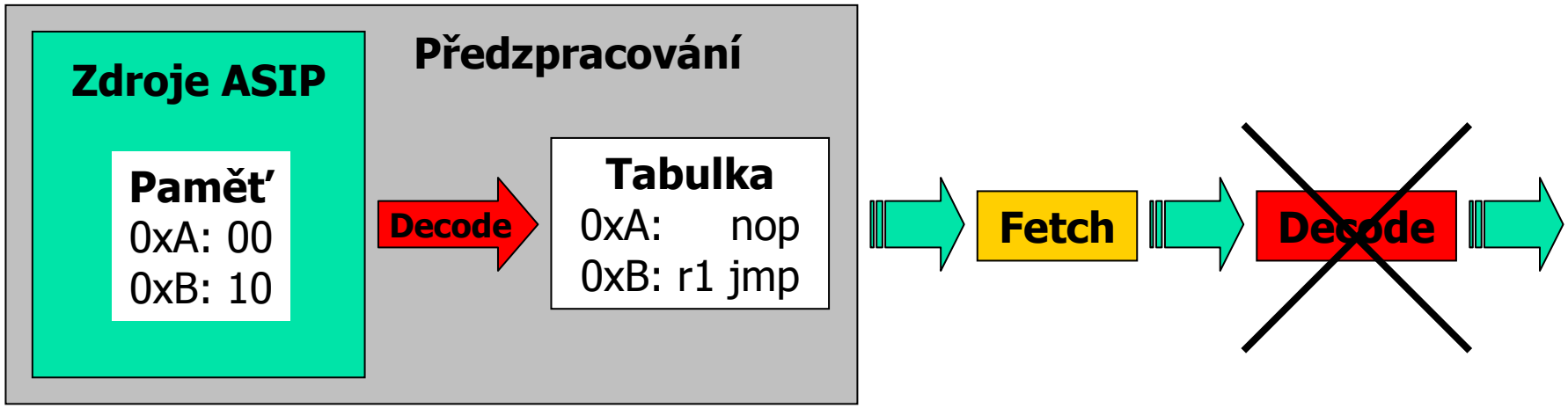
6. Interpretovaná a kompilovaná verze simulátoru

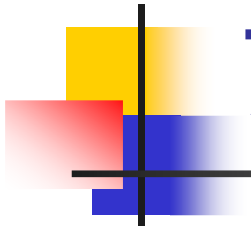
- Simulace založená na instrukcích i cyklech se může nacházet ve dvou variantách
- Interpretovaná simulace
  - Každá instrukce se vždy po načtení analyzuje
    - Získání operačních kódů, operandů etc.



6. Interpretovaná a kompilovaná verze simulátoru

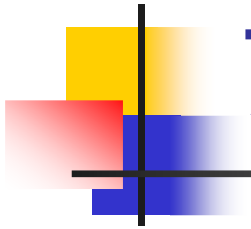
- Kompilovaná
  - Instrukce se analyzuje pouze jednou
  - Pro adresu instrukce v paměti se uloží operační kódy, operandy…





7. LISA - úvod

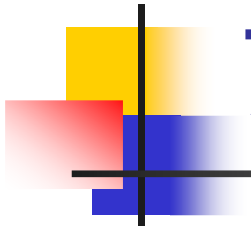
- **LISA** - Language for Instruction-Set Architecture
- Smíšená kategorie ADL
- Projekt Lissom
  - **ISAC** - Instruction Set Architecture C
    - Vychází z LISY
- LISA umožňuje specifikovat model ASIP tak, že je možno plně automaticky generovat všechny nástroje pro programování a simulaci ASIP
  - Simulátory založené na instrukcích i cyklech



7. LISA – struktura jazyka

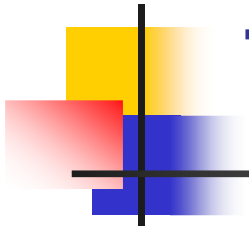
- Model LISY obsahuje deklaraci zdrojových prvků a popis operací hardwaru
  - Model architektury má dvě části

```
Description
    { ResourceSection }
    { Operation }
```



7. LISA – zdrojové prvky

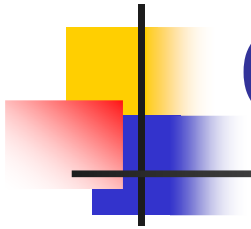
- Deklarace zdrojových prvků
  - Deklarované zdrojové prvky udržují stav programovatelné architektury ve formě uložené datové hodnoty
  - Sekce zdrojů definuje všechny zdroje modelu LISA jako je například
    - Zřetězené zpracování
    - Paměť
    - Registry …



7. LISA – zdrojové prvky

```
ResourceSection:
    { CDeclaration }
    'RESOURCE' '{'
      {
       CDeclaration      |
       ResourceElement   |
       MemoryElement     |
       MemoryMap         |
       PipelineResource  |
       AliasStatement
      }
    '}'
```

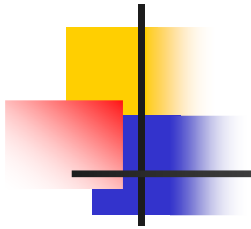
7. LISA – zdrojové prvky C/C++ deklarace



- Jsou podporovány deklarace podle konvencí jazyka C/C++
- Mohou se vyskytovat jak v sekci zdrojů, tak před ní

```
#define       U32    unsigned long
#define       UI     unsigned int
typedef       char*  data_t;
typedef       union{
   struct {
       UI     cpu_i          :8;
       UI     pwrd           :6;
       UI     sat            :1;
   }bit_sequence;
   U32 word;
}reg_t;
```

7. LISA – zdrojové prvky C/C++ deklarace



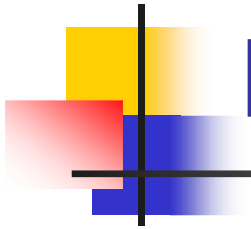
```
RESOURCE {
    int                 A;
    unsigned short      C;
    U32                 D;
    data_t              C;
    reg_t               etc;
}
```

7. LISA – zdrojové prvky
Jednoduché zdrojové prvky

```
ResourceSpecifier:
   'REGISTER'           |
   'CONTROL_REGISTER'   |
   'PROGRAM_COUNTER'    |
   'PIN'

RESOURCE {
    PROGRAM_COUNTER  int    pc;
    CONTROL_REGISTER int    ir;
    REGISTER         char   areg [0..15];
    PIN              int    status_bus;
}
```

7. LISA – zdrojové prvky
Linky zřetězení

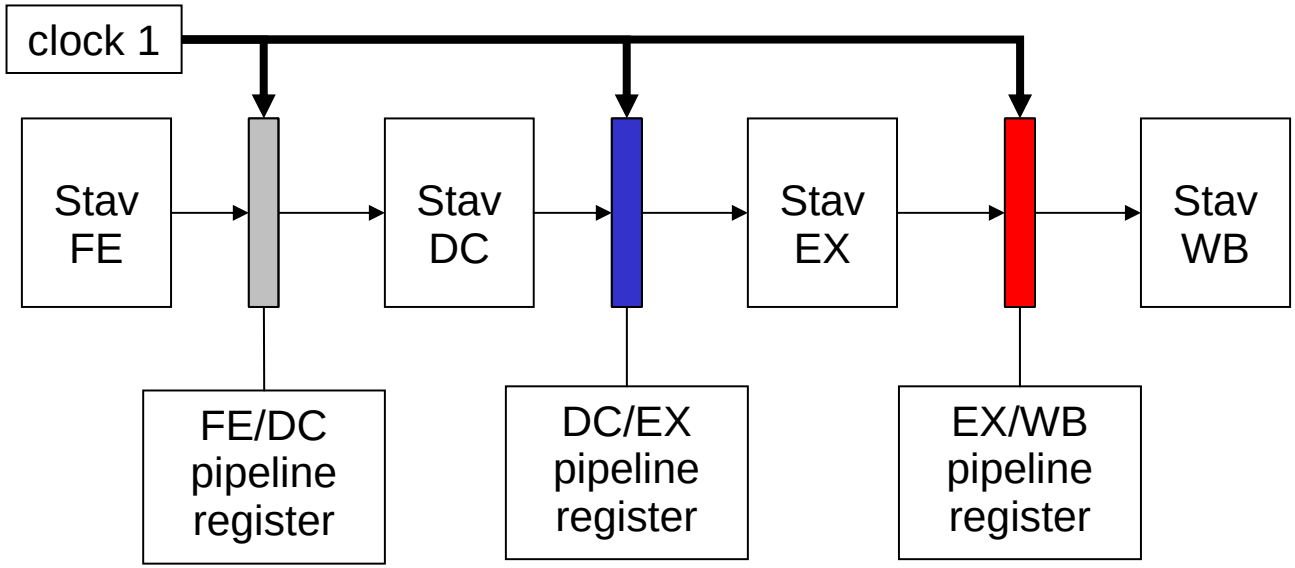


- Zlepšení průchodu instrukcí procesorem
- Zřetězené zpracování (pipeline) si musí pamatovat stavy instrukcí v různém stavu rozpracování
  - Kontext instrukce je udržován v pipeline registrech
- Registry musí být příslušné jistému typu pipeline - data jsou v registrech postupně předávána
- S registry lze pracovat v C/C++ kódu - přístup přes speciální funkce

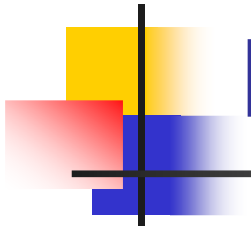
```
PIPELINE pipe = { FE; DC; EX; WB };}
PIPELINE_REGISTER IN pipe {
    int instruction_register;
    Short program_counter;
    REGISTER bit[24] src1, src2, dest;
}
```

7. LISA – zdrojové prvky
Linky zřetězení

- Zřetězené zpracování sestává ze čtyř stavů: fetch (FE), decode (DC), execute (EX),writeback (WB)
- Přístup k registru - **PIPELINE_REGISTER**(pipe,DC/EX).src1



7. LISA – zdrojové prvky Paměti



- Jazyk umožňuje popisovat ideální paměti (MEMORY), stejně jako neideální komponenty RAM, CACHE a sběrnice (BUS)
- Ideální pamět - ideální paměť je pole pevné délky složené z prvků definované velikosti (bit-width)
  - Prvky jsou přístupné paměťovou adresou (index do pole)
  - Paměť může obsahovat proveditelný kód
  - Paměť se může skládat z banků (banked memory)

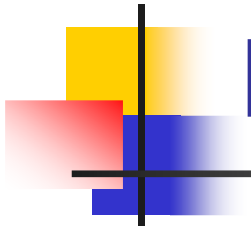
```
MEMORY unsigned int pmem {
   BLOCKSIZE (32, 32);
   SIZE(0x10000);
   FLAGS      (R|W|X);
};
```

7. LISA – zdrojové prvky Paměti

```
RAM unsigned char   dmem [4]{
   BLOCKSIZE (8, 8);
   SIZE(0x400000);
   FLAGS      (R|W);
};
```

- Paměť *pmem* obsahuje 0x10000 prvků typu unsigned int
  - Povolený rozsah indexů je od 0x0 do 0xffff
- Druhá paměť typu RAM *dmem* sestává ze čtyř paměťových banků, každý o velikosti 0x400000 bytů

7. LISA – zdrojové prvky
Přístup do paměti



- **Čistě funkcionální** - paměť je modelována jako čisté pole C
  - K paměti je přistupováno operátorem [ ], data jsou vracena bezprostředně a není shromažďována žádná statistika
- **Přesné počítání cyklů -** přístup pro sběrnice, RAM a cache
  - Užívána malá množina funkcí rozhraní - data jsou vracena bezprostředně, je vraceno množství skrytých cyklů, je shromažďována statistika
  - Je povoleno používat operátor [ ] - dovoluje hladkou migraci z čistě funkcionálního modelu
- **Založeno na cyklech –** je nejpřesnější modelovací metoda
  - Podává zprávu o omezeném počtu paměťových/sběrnicových portů, přístup do paměti z různých fází linky zřetezení

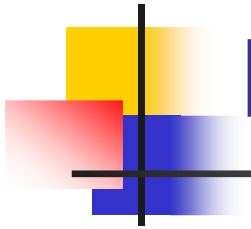
7. LISA – zdrojové prvky
Přístup do paměti – přesné počítání cyklů

```
RESOURCE {
    RAM       int    dmem    { … };
    CACHE     int    dcache  { CONNECT(dmem); … };
}
OPERATION ADDM {
    BEHAVIOR {
        dcache.read(addr,R);      // R=memory@addr
        R = R + operand;          // do add operation
        dcache.write(addr,R);     // memry@addr = R
    }
}
```

7. LISA – zdrojové prvky
Přístup do paměti – založený na cyklech

```
OPERATION LDR1 IN pipe.EX1 {
    BEHAVIOR {
        int s = dcache.request_read(addr,R);
        if ( s != MA_OK ) {
            PIPELINE_REGISTER(pipe,ID).stall()
            PIPELINE_REGISTER(pipe,IF).stall();
        }
    }
    ACTIVATION {
      if ( s != MA_OK ) {
        LDR1;
      }
    }
}
```

7. LISA – zdrojové prvky
Mapování paměti

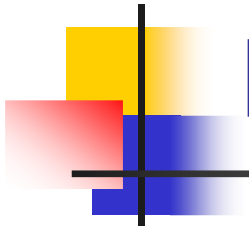


- Pro spuštění simulátoru je nezbytné zobrazení adresového prostoru procesoru nad definovanými paměťmi
- Zobrazení musí být jednoznačné
- Existuje jedno nebo více mapování paměti
  - Jedno implicitní mapování
- Simulátor může přepínat mezi různými mapováními paměti
- Každá položka mapování paměti zobrazuje obvykle koherentní rozsah paměťových adres do paměťového zdroje
- Mapování obsahuje identifikaci paměťového zdroje (CACHE, RAM, MEMORY) a popisuje mapování vybraných bitů do indexů paměťového zdroje

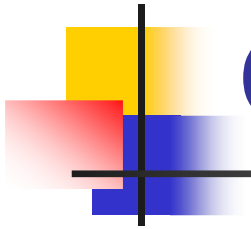
7. LISA – zdrojové prvky
Mapování paměti

```
MEMORY   char      memory1 {
   BLOCKSIZE       (8, 8);
   SIZE            (0x800000);
};
MEMORY   int       memory2 {
   BLOCKSIZE       (32, 32);
   SIZE            (0x100000);
};
MEMORY   char      memory3 {
   BLOCKSIZE       (8, 8);
   SIZE            (0x800000);
};
MEMORY_MAP         default {
   RANGE(0x000000,0x7fffff) -> memory1[(31..0)];
   RANGE(0x800000,0x7bffff) -> memory2[(31..2)];
};
MEMORY_MAP          blocked {
   RANGE(0x000000,0x7fffff), PAGE (0) -> memory1[(31..0)];
   RANGE(0x000000,0x7fffff), PAGE (1) -> memory3[(31..0)];
};
```

7. LISA – zdrojové prvky
Mapování paměti



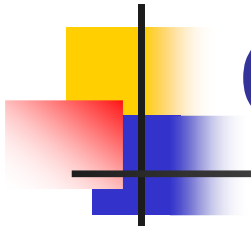
- Jsou definovány tří různé paměti - dvě RAM paměti velikosti 0x800000 bytů, jedna RAM velikosti 0x400000
- Modelovaná architektura používá 32 bitový adresový prostor a bajtové adresování
- První implicitní mapování adres zobrazuje adresový rozsah od 0 do 0xbfffff do pamětí *memory1* a *memory2*
- Druhé mapování paměti popisuje blokové adresování
- Je použito dvou paměťových stránek se stejným adresovým rozsahem pro zobrazení do pamětí *memory1* a *memory3*
  - Užíváno pro signálové procesory, kde oddělené stránky jsou užity pro paměť programů, dat a periferií



7. LISA – operační část
Operace

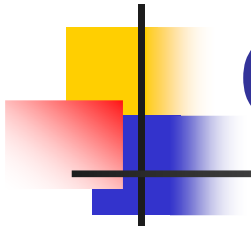
- Operace je základní objekt v jazyku LISA
- Obsahuje kolekci popisů různých vlastností systému
  - Vlastnosti == model chování, časování, atp. z ADL
- Operace je popsána v následujících sekcích
  - Sekce **DECLARE**
    - Lokální deklarace identifikátorů, skupin operací a odkazů na jiné operace
  - Sekce **CODING**
    - Popisuje binární obraz instrukčního slova

7. LISA – operační část
Operace



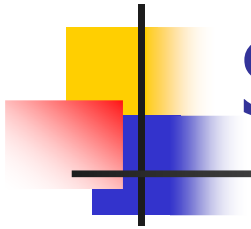
- Sekce **SYNTAX**
  - Mnemonika a jiné syntaktické komponenty jazyka assembleru jako jsou operandy a prováděcí módy
- Sekce **BEHAVIOR** a **EXPRESSION**
  - Komponenty modelu chování (popis semantiky instrukcí)
- V sekci **ACTIVATION**
  - Je definováno časování jiných operací vzhledem k popisované operaci

7. LISA – operační část
Operace



- Prováděné operace vedou systém do nového stavu během simulace
- Operace mohou být buď atomické nebo se instrukce skládají z jiných operací
  - Model podporuje hierarchické strukturování operací
    - Graf je vytvářen odkazy na jiné operace
      - Je povoleno ve všech sekcích popisu operací
      - Kompletace nadřazených definic operací (neterminálních) je provedena pomocí odkazovaných neterminálů nižší úrovně a terminálů

7. LISA – operační část
Sekce DEKLARE

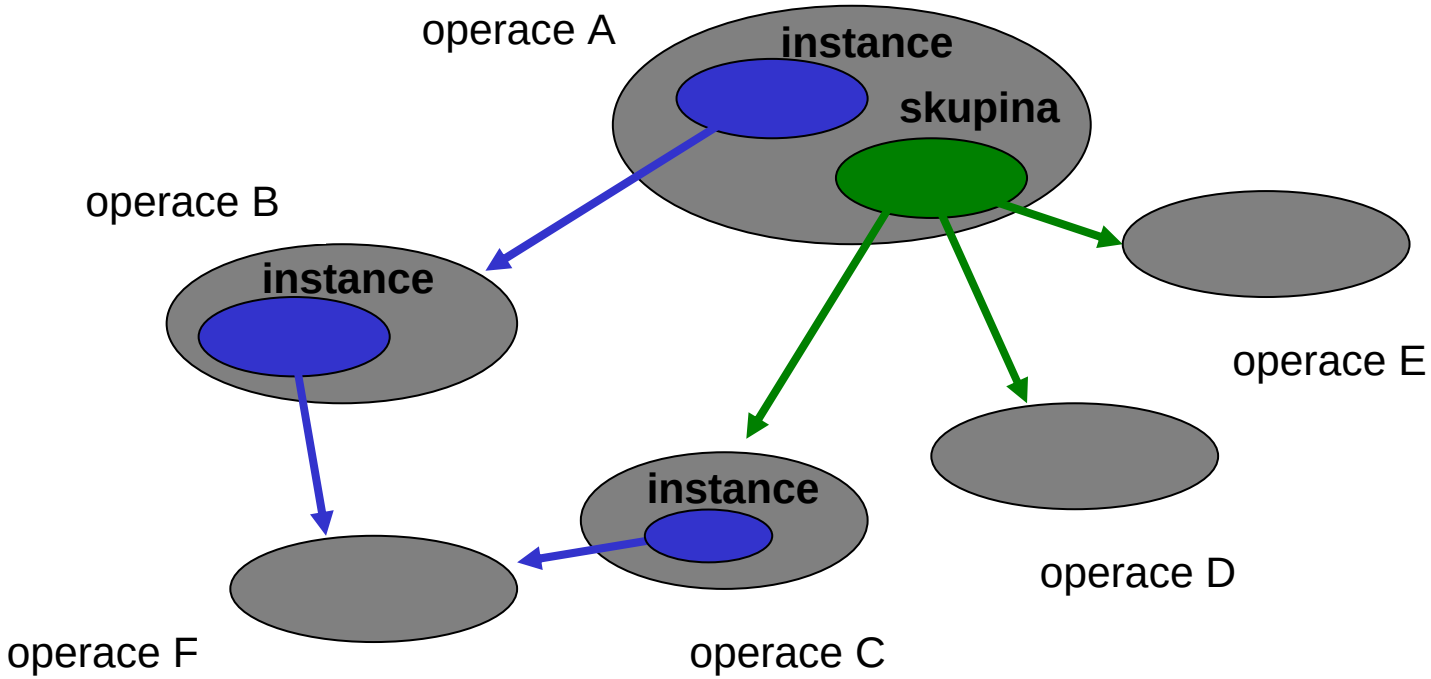


- Deklarace definuje lokální deklaraci platnou v rámci všech sekcí aktuální operace
- Máme tři typy deklarací
  - Instance (**INSTANCE**)
  - Skupiny (**GROUP**)
  - Návěští (**LABEL**)

```
OPERATION A {
    DECLARE  { … }
}
```

7. LISA – operační část
Instance a skupiny

- Instance – odkazy na jiné operace
- Skupiny – varianty; sdružují operace, které mohou být užity ve stejném kontextu



7. LISA – operační část
Instance a skupiny

- Všechny prvky **skupiny** musejí poskytovat tytéž definice (např. sekce ASSEMBLER a CODING), pokud jsou použity v nadřazené definici

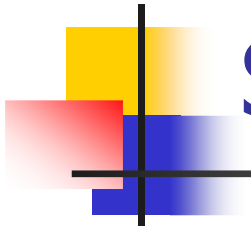
```
OPERATION A {
   DECLARE {
      INSTANCE B;
      GROUP group = { C || D || E };
   }
   CODING    { … }
   SYNTAX    { … }
   BEHAVIOR  { … }
}
```

7. LISA – operační část
Návěští

- Návěští umožňuje pojmenovávat úseky textu assembleru nebo binárního kódu
- V deklaraci jsou vyjmenována jména pro toto použití

```
OPERATION ADD {
   DECLARE {
      LABEL index;
      GROUP src, dest = { register };
   }
   CODING    {index=0bx[14] 0b1 src dest }
   SYNTAX    { "ADD" dest "," src "," index=#U }
   BEHAVIOR { dest = src + index; }
}
```

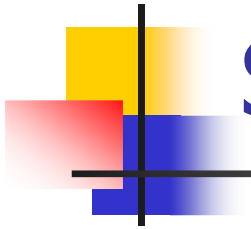
7. LISA – operační část
Sekce CODING



- Sekce CODING je tvořena zřetězenými elementy, který tvoří posloupnost terminálních a neterminálních symbolů
- Terminální symbol tvoří bitové pole
- Užitím reference se neterminálním symbolem odkazujeme na sekci CODING jiné operace

```
OPERATION register {
   CODING { 0bx[4] }
}
OPERATION decode {
    DECLARE {
      GROUP src1, src2, dest = { register };
    }
    CODING { 0b0011 src1 src2 dest 0b1[3] }
}
```

7. LISA – operační část
Sekce SYNTAX

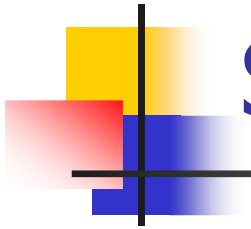


- Sekce SYNTAX definuje
  - Výstup překladače jazyka C, který transformuje C jako vyšší programovací jazyk do assemblerovského kódu
  - Informaci pro transformaci assemblerovského textu na binární (assembler)
  - Informaci pro transformaci binárního textu na assemblerovský (disassembler)
- Textová reprezentace instrukcí na assemblerovské úrovni
- Prvek popisu je buď terminální nebo neterminální symbol
- Terminály jsou řetězce nebo návěští reprezentující čísla, která přímo odpovídají kódovaným prvkům

7. LISA – operační část
Sekce SYNTAX

```
OPERATION ADD {
   DECLARE {
      LABEL index;
      GROUP src, dest = { register };
   }
   CODING    { 0b1 src dest index=0bx[14] }
   SYNTAX    { "ADD" dest "," src "," index=#U }
}
```

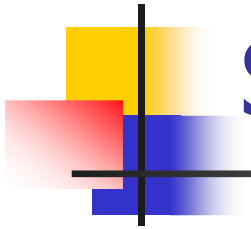
7. LISA – operační část
Sekce BEHAVIOR



- Popis chování instrukce v jazyce C/C++
- Funkce komunikuje s okolím výhradně přes globální zdroje
- Volání chování jiné operace se děje uvedením jiné instance, reference nebo skupiny

```
OPERATION ADD {
    CODING { 0b101011 mode immediate=0bx[10] }
    BEHAVIOR {
        status();ALU();mode();
        R1 = do_ADD(immediate);
    }
}
```

7. LISA – operační část
Sekce EXPRESSION



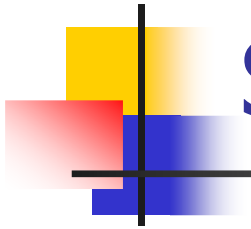
- V sekci se definuje výraz, který vrací buďto zdrojový prvek nebo numerický výraz
- Objekt ve výrazu poskytuje operandy k sekci BEHAVIOR příslušné operace
- Výrazy jsou velmi užitečné pro specifikaci platných registrů, které mají být zpřístupňovány některou instrukcí

```
OPERATION register {
   DECLARE{ LABEL index;}
   CODING {index = 0bx[4]}
   SYNTAX {"A" index}//jméno registru je A0..A15
   EXPRESSION {R[index]}//identifikátor pro přístup reg.
}
```

7. LISA – operační část
Sekce EXPRESSION

```
OPERATION addr {
     DECLARE {
          LABEL position;
     }
     CODING     {position=0bx[16]} // kódování sestává ze 16b
     SYNTAX     {position=#U } // číslo bez znaménka v adrese
     EXPRESSION {position} // vrací číslo z kódování
}
OPERATION load {
     DECLARE {
          GROUP dest = {register};
          INSTANCE addr;
     }
     CODING   {0b0100101011 addr dest }
     SYNTAX   {dest "=" "DM(" addr ")" }
     BEHAVIOR { dest=memory[addr]; }
}
```

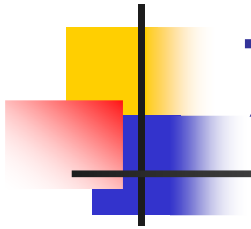
7. LISA – operační část
Sekce ACTIVATION



- Sekce aktivace je seznamem operací, které se mají provést jako následující krok
- Operace mohou být aktivovány užitím skupiny, instance nebo reference
- Operace jsou prováděny podle jejich přiřazení lince zřetězení
- Aktivované operace obvykle aktivují další operace, které produkují aktivační řetěz

7. LISA – operační část
Sekce ACTIVATION

```
OPERATION fetch IN pipe.FE {
     DECLARE { INSTANCE decode; }
     BEHAVIOR { instruction_register = prog_mem[pc];}
     ACTIVATION { decode }
}
OPERATION decode IN pipe.DC {
     DECLARE { INSTANCE add, writeback; }
     ACTIVATION { add, writeback }
}
OPERATION add IN pipe.EX {
     DECLARE { GROUP src1, src2, dest={ register; }
     BEHAVIOR { result = src1 + src2; }
}
OPERATION writeback IN pipe.WB {
     DECLARE { REFERENCE dest; }
     BEHAVIOR { dest = result; }
}
```



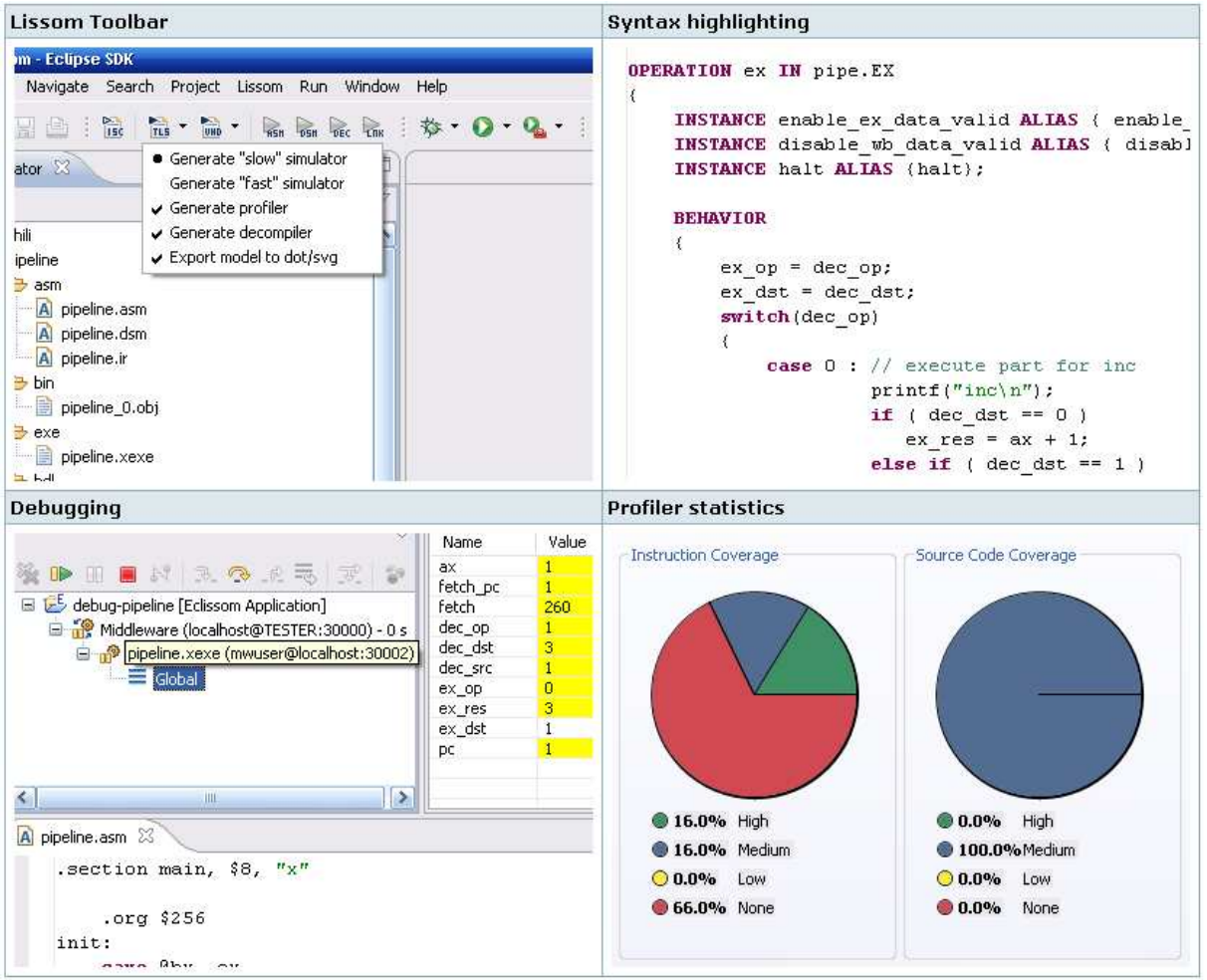
7. LISA – ARM 7

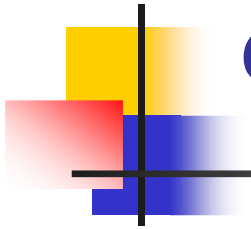
```
OPERATION main
{
    DECLARE {
      INSTANCE Decode_Instruction, Inerrupt_detection;
    }
    BEHAVIOR {
        // provedení instrukce
        Decode_Instruction();
        // Přerušení bude aktivní u další instrukce
        // -> tj. je testováno po provedení instrukce
        Interrupt_Detection();
        // Zvýšení programového čitače
        PC += 4; // všechny instrukce mají stejnou šířku
    }
}
```

8. Nástroje pro návrh ASIP

www.fit.vutbr.cz/research/groups/lissom

8. Nástroje pro návrh ASIP





9. Závěr

Děkuji za pozornost