Vysoké učení technické v Brně

Fakulta informačních technologií

Semestrální práce z předmětu VPD: Vnitřní model jazyka ISAC

Roman Lukáš

2004

Obsah

1. Úvod 3

2. Vnitřní modely jednotlivých částí 3

2.1 Model fyzické struktury počítače 3
2.1.1 Mapování paměti 6
2.2 Model popisu operací a chování 9
2.2.1 Model popisu hierarchie operací 9
2.2.2 Model popisu sekce kódu 12
2.2.2.1 Řídicí struktura 12
2.2.2.2 Atomární příkazy 15
2.2.2.2.1 Sekce Coding 15
2.2.2.2.2 Sekce Assembler 17
2.2.2.2.3 Sekce Behavior 19

1. Úvod

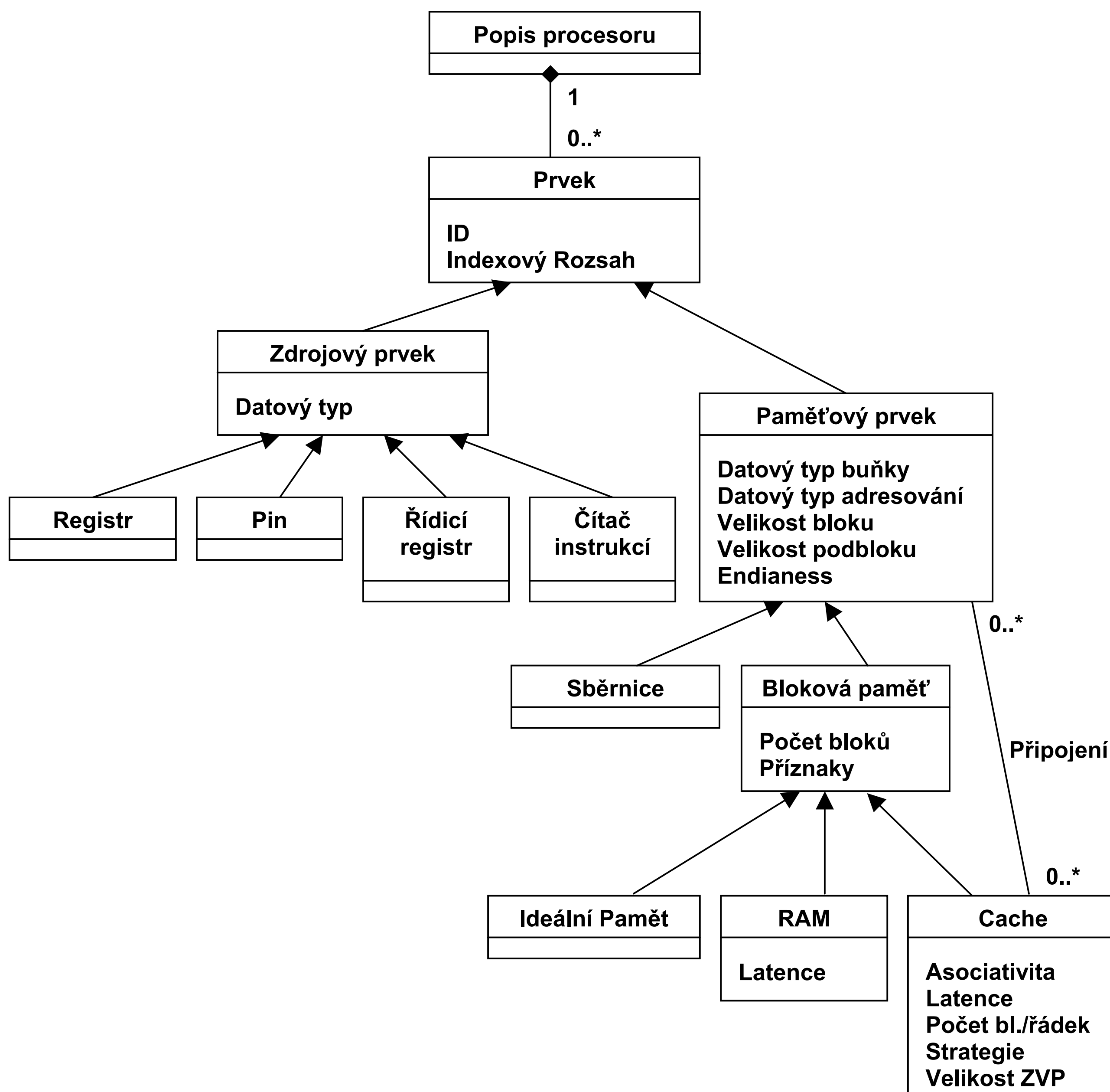
ISAC je speciální programovací jazyk pro strukturu procesoru, syntaxi operací procesoru, jejich kódování a chování. Cílem této práce je navrhnout vnitřní model, kterým může být vhodně reprezentován každý program napsaný v tomto jazyce.

Vnitřní model ISAC se skládá ze dvou základních částí. V první části je popsán model fyzické struktury procesoru. V druhé části je popsán model operací, který popisuje jak syntaxi jednotlivých operací, tak jejich kódování do binárního souboru. Dále je popsáno chování operací, které je potřeba pro správnou činnost simulátoru. Obě tyto části mohou být modelovány odděleně.

2. Vnitřní modely jednotlivých částí

2.1 Model fyzické struktury počítače

Následující model ukazuje základní hardwarové části procesoru:



Modelovaný počítač obsahuje prvky. Každý z těchto prvků má svůj identifikátor. Dalším atributem je indexový rozsah, který specifikuje pro pole prvků rozsah indexových hodnot. Tyto prvky můžeme rozdělit na zdrojové a paměťové. Zdrojovým prvkem může být registr, pin, řídící registr nebo čítač instrukcí. U těchto prvků je specifikován pouze datový typ. Dále může počítač obsahovat paměťové prvky, které se dále specifikují na sběrnice a blokové paměti, přičemž blokovými pamětmi máme namysli ideální paměť, paměť RAM a cache. Speciálně cache může být připojena k jiné paměti, což je modelováno vazbou „připojení“. Parametry paměťových prvku jsou následující:

- **Datový typ buňky** – Určuje, jakého typu jsou jednotlivé dílčí položky v paměti
- **Datový typ adresování** – Typ adresování použitý v paměti (musí být definován v typedef). Pokud není zadán uživatelem, doplní se na `unsigned longint`.
- **Velikost bloku** – Udává bitovou velikost paměti.
- **Velikost podbloku** – Pokud není zadána uživatelem, doplní se pomocí funkce `sizeof`.
- **Endianess –** nabývá hodnot LITTLE a BIG.

Speciální atributy pro blokové paměti:

- **Počet bloků** – Definuje počet bloků paměti.
- **Příznaky** – 3-bitové slovo, kde první bit specifikuje, zda je paměť čitelná, druhý specifikuje, zda je pamět zapisovatelná a třetí, zda může obsahovat proveditelný kód. Pokud není zadán uživatelem, nastaví všechny hodnoty na true.

Speciální atribut pro RAM a Cache:

- **Latence** – Specifikuje latenci pro operaci čtení a zápisu (první a druhý parametr). Pokud není zadán uživatelem, doplní se na hodnotu (1, 1).

Speciální atributy pouze pro Cache:

- **Asociativita** – specifikuje asociativitu cache, tj. počet cest pro set-asociativní cache. Pokud není zadán uživatelem, doplní se na hodnotu 1.
- **Počet bloků/řádek** - definuje velikost volitelné zápisové vyrovnávací paměti (v řádcích cache)
- **Strategie** – 3-bitové slovo, kde první bit specifikuje, zda je WA_Policy nastaveno na NEVER / ALWAYS, druhý bit specifikuje, zda je WB_Policy nastaveno na NEVER / ALWAYS a třetí bit, zada je LRU_Policy nastaveno na LRU. Pokud není zadán uživatelem, nastaví hodnoty postupně v tomto pořadí: NEVER, NEVER, LRU.
- **Velikost ZVP** - definuje velikost volitelné zápisové vyrovnávací paměti (v řádcích cache)

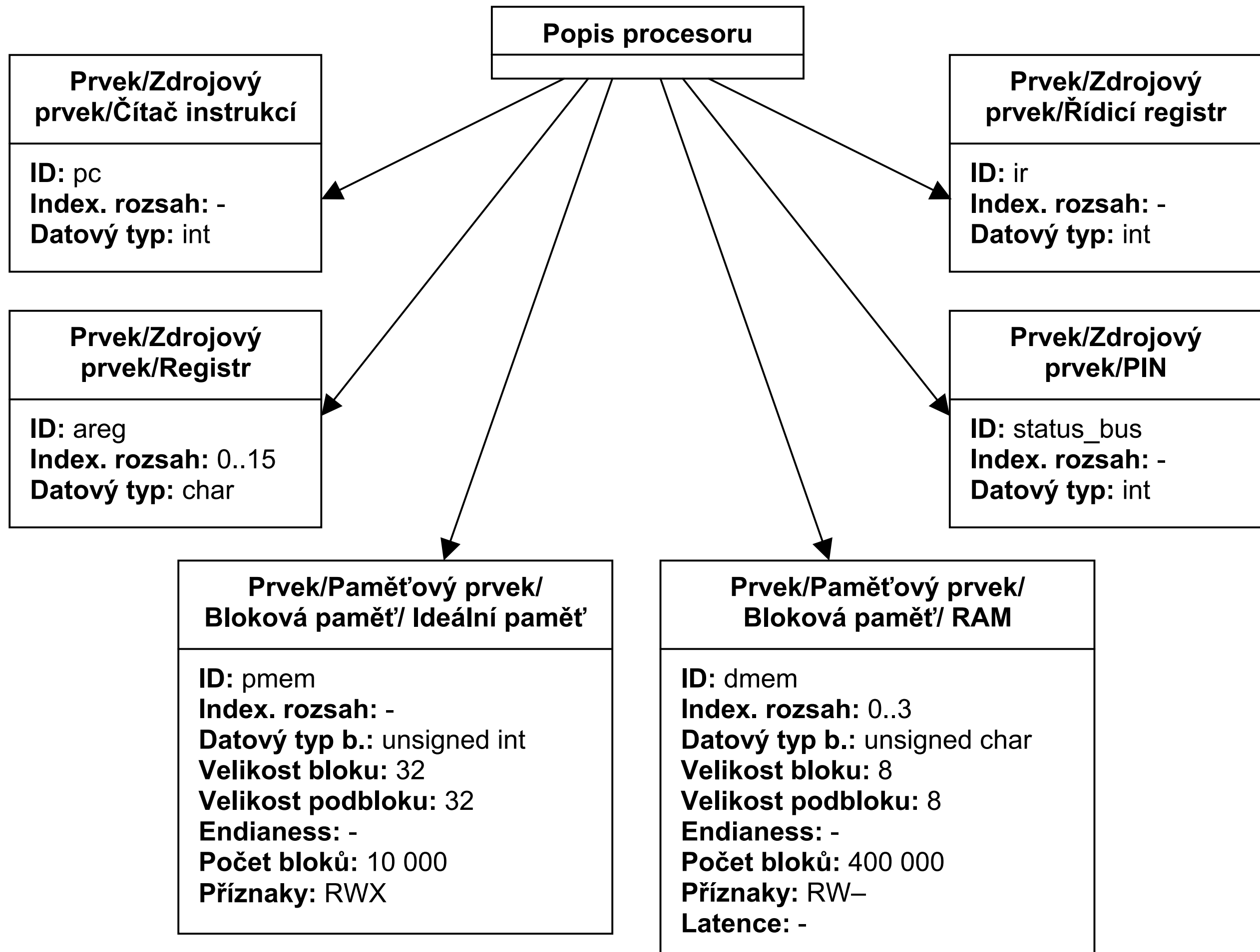
Příklad:

Uvažujme následující kód:

```
RESOURCE {
            PROGRAM_COUNTER   int               pc;
            CONTROL_REGISTER  int               ir;
            REGISTER          char              areg [0..15];
            MEMORY            unsigned int      pmem {
                  BLOCKSIZE   (32, 32);
                  SIZE        (0x10000);
                  FLAGS       (R|W|X);
                  };
```

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

Model tohoto kódu je následující:



Odpovídající část XML kódu:

```
<DESCRIPTION>
      <PROGRAM_COUNTER>
            <ID> pc </ID>
            <DATE_TYPE> int </DATE_TYPE>
      </PROGRAM_COUNTER>
      <CONTROL_REGISTER>
            <ID> ir </ID>
            <DATE_TYPE> int </DATE_TYPE>
      </CONTROL_REGISTER>
      <REGISTER>
            <ID> areg </ID>
            <INDEX_RANGE> 0-15 </INDEX_RANGE>
            <DATE_TYPE> char </DATE_TYPE>
      </REGISTER>
```

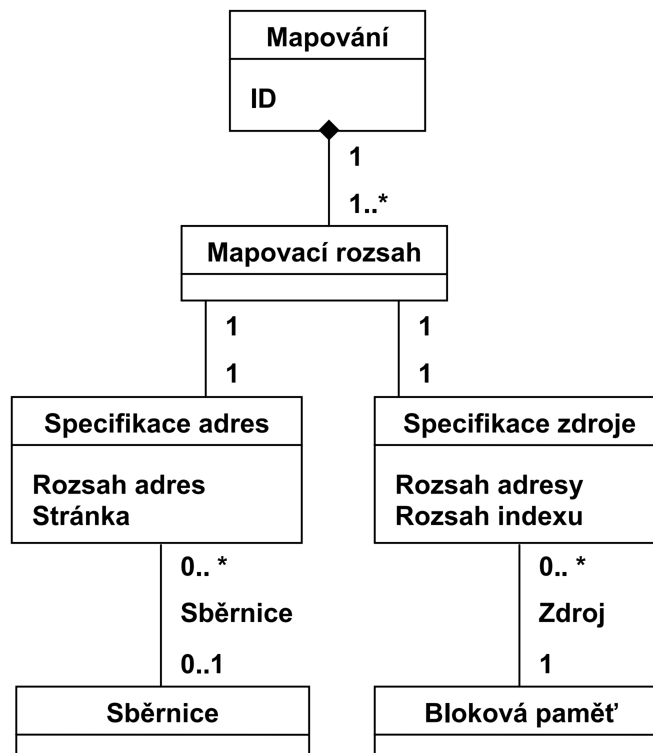
```
	<MEMORY>
		<ID> pmem </ID>
		<DATE_TYPE> unsigned int </DATE_TYPE>
		<BLOCK_SIZE> 32 </BLOCK_SIZE>
		<SUBBLOCK_SIZE> 32 </SUBBLOCK_SIZE>
		<COUNT_OF_BLOCKS> 10000 </COUNT_OF_BLOCKS>
		<FLAGS> RWX </FLAGS>
	</MEMORY>
	<RAM>
		<ID> dmem </ID>
		<DATE_TYPE> unsigned char </DATE_TYPE>
		<INDEX_RANGE> 0-3 </INDEX_RANGE>
		<BLOCK_SIZE> 8 </BLOCK_SIZE>
		<SUBBLOCK_SIZE> 8 </SUBBLOCK_SIZE>
		<COUNT_OF_BLOCKS> 400000 </COUNT_OF_BLOCKS>
		<FLAGS> RW- </FLAGS>
	</RAM>
	<PIN>
		<ID> status_bus </ID>
		<DATE_TYPE> int </DATE_TYPE>
	</PIN>
</DESCRIPTION>
```

Konec příkladu

Poznámka: Hodnoty, které nejsou v jazyce ISAC definovány, nejsou v XML-kódu nijak specifikovány.

2.1.1 Mapování paměti

Následující model ukazuje, jak budeme ve vnitřní struktuře modelovat mapování:



Mapování rozsahu adres může být rozděleno na několik dílčích mapovacích rozsahů, které mohou být obecně mapovány do různých pamětí. Jednotlivé dílčí mapovací rozsahy jsou definovány samotným

rozsahem mapovacích adres (třída: Specifikace adres) a specifikací zdroje a jeho skutečných adres, na které je mapování zobrazeno (třída: Specifikace zdroje).

Třída Specifikace adres obsahuje atribut rozsah, který specifikuje samotný rozsah adres mapovatelný do skutečné paměti, který může být upřesněn i stránkou paměti. Speciálně pro mapování sběrnic je specifikována aktuální sběrnice.

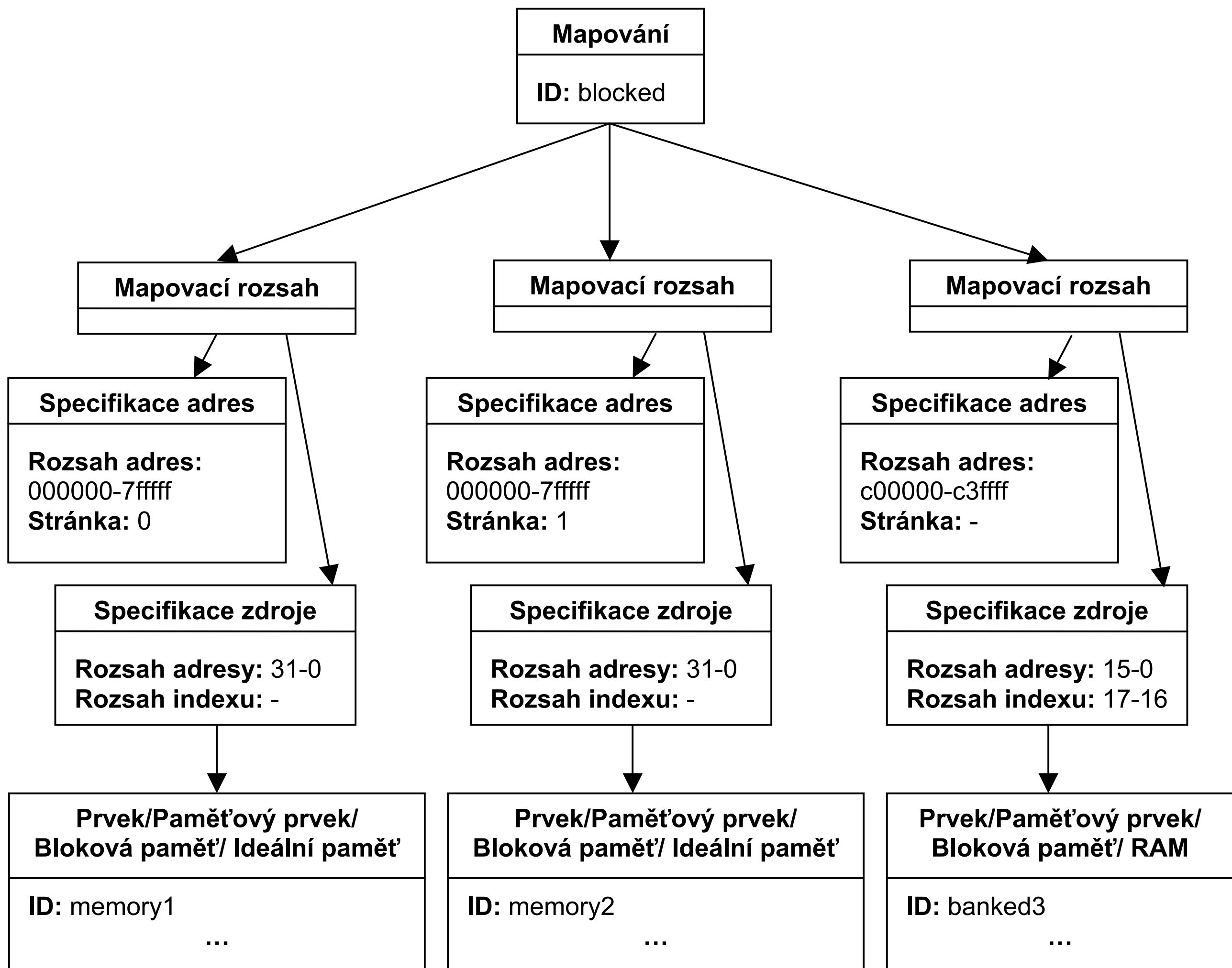
Třída Specifikace zdroje se hlavně odkazuje na blokovou paměť, do které je mapování prováděno. Atribut „Rozsah adresy“ popisuje rozsah adresového slova ze kterého bude vypočítána skutečná adresa zdroje. Atribut „Rozsah indexu“ je použit pouze pro pole pamětí a popisuje rozsah adresového slova, ze kterého bude určen index dané paměti.

Příklad:

Uvažujme následující kód:

```
RESOURCE {
      MEMORY char memory1 {…}
      MEMORY int  memory2 {…}
      RAM    char banked3 [4] {…};
      MEMORY_MAP        blocked {
            RANGE(0x000000,0x7fffff), PAGE (0) -> memory1 [(31..0)];
            RANGE(0x000000,0x7fffff), PAGE (1) -> memory2 [(31..0)];
            RANGE(0xc00000,0xc3ffff) -> banked3 [(17..16)][(15..0)];
            };
      }
```

Model tohoto kódu je následující:



Odpovídající část XML kódu:

```
<MAPING>
	<MAPING_RANGE>
		<ADDRESS_SPEC>
			<RANGE> 000000-7fffff </RANGE>
			<PAGE> 0 </PAGE>
		</ADDRESS_SPEC>
		<RESOURCE_SPEC>
			<ADRESS_RANGE> 31-0 </ADRESS_RANGE>
			<SOURCE> memory1 </SOURCE>
		</RESOURCE_SPEC>
	</MAPING_RANGE>
	<MAPING_RANGE>
		<ADDRESS_SPEC>
			<RANGE> 000000-7fffff </RANGE>
			<PAGE> 1 </PAGE>
		</ADDRESS_SPEC>
		<RESOURCE_SPEC>
			<ADRESS_RANGE> 31-0 </ADRESS_RANGE>
			<SOURCE> memory2 </SOURCE>
		</RESOURCE_SPEC>
	</MAPING_RANGE>
	<MAPING_RANGE>
		<ADDRESS_SPEC>
			<RANGE> c00000-c3ffff </RANGE>
		</ADDRESS_SPEC>
		<RESOURCE_SPEC>
			<ADRESS_RANGE> 15-0 </ADRESS_RANGE>
			<INDEX_RANGE> 17-16 </ INDEX_RANGE>
			<SOURCE> banked3 </SOURCE>
		</RESOURCE_SPEC>
	</MAPING_RANGE>
</MAPING>
```

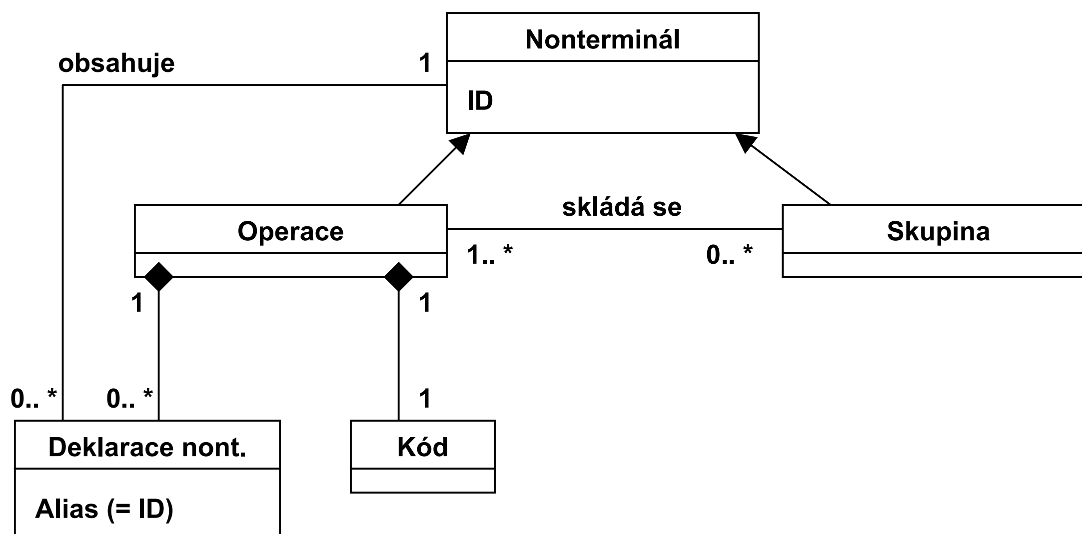
Konec příkladu

2.2 Model popisu operací a chování

V této kapitole jsou navrženy modely pro popis hierarchického zanoření operací, jejich syntaxe a kódování v binárním kódu. Tato kapitola je rozdělena do několika logických celků, kde v každém z nich je popsána jistá logická struktura.

2.2.1 Model popisu hierarchie operací

Následující model ukazuje základní vztah mezi operacemi a skupinami, pomocí kterého může být vytvořena jistá hierarchie operací:



Nonterminál je abstraktní třída, která je dále specializována na operaci nebo skupinu. Skupina se dále skládá z jedné nebo více operací a naopak každá operace může být součástí několika skupin, popřípadě i žádné.

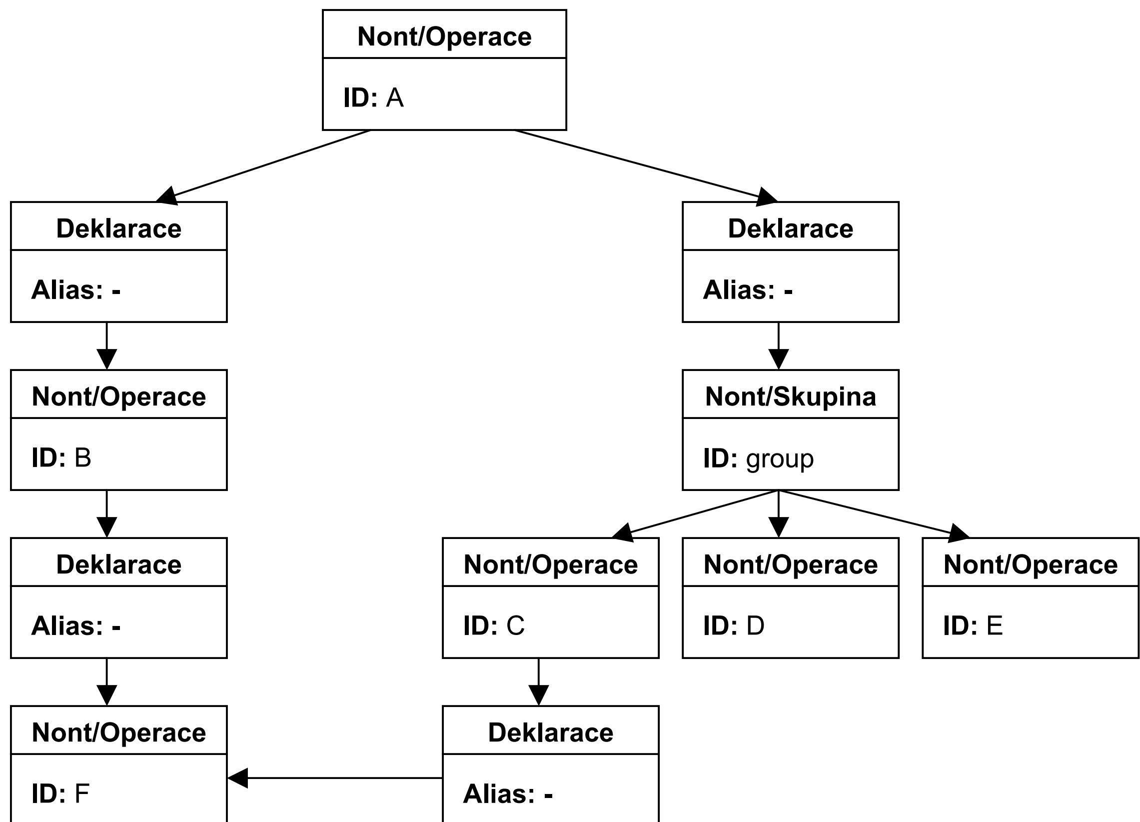
Operace se dále skládá z lokálních deklarací a kódu. Deklarován je vždy nonterminál. Pokud je potřeba deklarovat více stejných nonterminálů, například potřebujeme pojmenovat více registrů, budou tyto jednotlivé názvy uvedeny jako atribut Alias ve třídě deklarace. Sekce kódu bude modelována později.

Příklad:

Uvažujme následující kód:

```
GROUP group = C,D,E;
OPERATION A {
     INSTANCE B, group;
     ...
  }
  OPERATION B {
     INSTANCE F;
     ...
  }
  OPERATION C {
     INSTANCE F;
     ...
  }
  OPERATION D { … }
  OPERATION E { … }
  OPERATION F { … }
```

Model tohoto kódu je následující:



Odpovídající část XML kódu:

```
<GROUP>
      <ID> group </ID>
      <MEMBERS>
            <MEMBER> C </MEMBER>
            <MEMBER> D </MEMBER>
            <MEMBER> E </MEMBER>
      </MEMBERS>
</GROUP>
<OPERATION>
      <ID> A </ID>
      <DECLARATIONS>
            <DECLARATION>
                  <NONTERMINAL> B </NONTERMINAL>
            </DECLARATION>
            <DECLARATION>
                  <NONTERMINAL> group </NONTERMINAL>
            </DECLARATION>
      </DECLARATIONS>
</OPERATION>
<OPERATION>
      <ID> B </ID>
      <DECLARATIONS>
            <DECLARATION>
                  <NONTERMINAL> F </NONTERMINAL>
            </DECLARATION>
      </DECLARATIONS>
</OPERATION>
```

```
<OPERATION>
	<ID> C </ID>
	<DECLARATIONS>
		<DECLARATION>
			<NONTERMINAL> F </NONTERMINAL>
		</DECLARATION>
	</DECLARATIONS>
</OPERATION>
<OPERATION>
	<ID> D </ID>
</OPERATION>
<OPERATION>
	<ID> E </ID>
</OPERATION>
<OPERATION>
	<ID> F </ID>
</OPERATION>
```

Konec příkladu

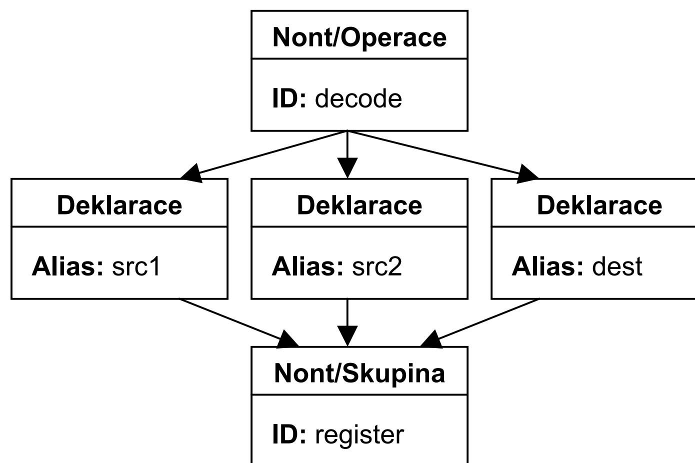
Příklad:

Uvažujme následující kód:

```
OPERATION decode {
  INSTANCE register ALIAS  { src1, src2, dest } ;

    ...
    }
```

Model tohoto kódu je následující:

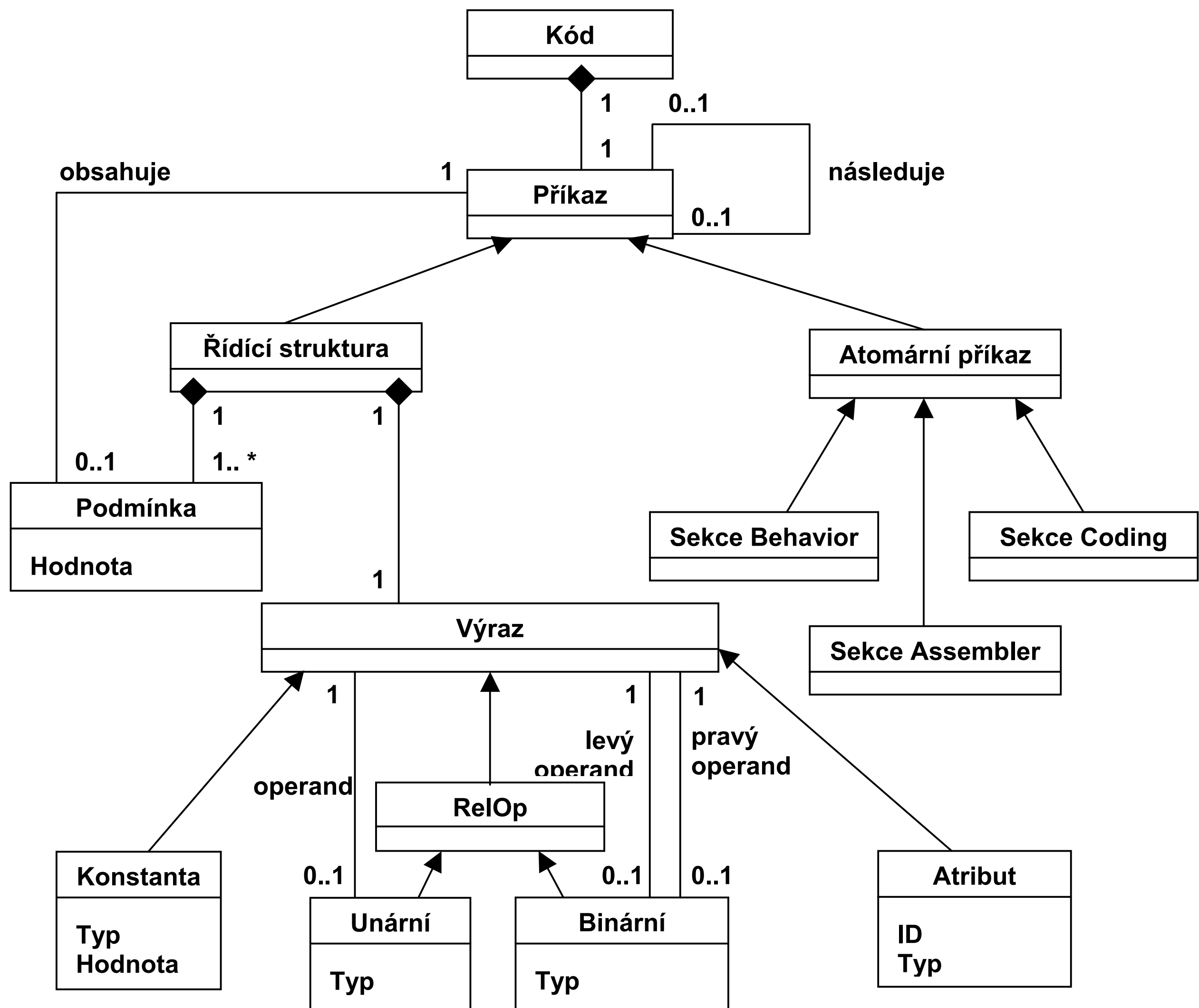


Odpovídající část XML kódu:

```
<OPERATION>
	<ID> decode </ID>
	<DECLARATIONS>
		<DECLARATION>
			<NONTERMINAL> register </NONTERMINAL>
			<ALIAS> scr1 </ALIAS>
		</DECLARATION>
		<DECLARATION>
			<NONTERMINAL> register </NONTERMINAL>
			<ALIAS> scr2 </ALIAS>
		</DECLARATION>
		<DECLARATION>
			<NONTERMINAL> register </NONTERMINAL>
			<ALIAS> dest </ALIAS>
		</DECLARATION>
	</DECLARATIONS>
</OPERATION>
```

2.2.2 Model popisu sekce kódu

Následující model ukazuje základní strukturu kódu operace a strukturu jednotlivých příkazů:



Základními elementy kódu je sekvence příkazů. Příkazem může být buď řídící struktura nebo atomární příkaz.

2.2.2.1 Řídicí struktura

Každá řídicí struktura obsahuje právě jeden výraz. Až po vyhodnocení tohoto výrazu se rozhodne, který příkaz bude vykonán. Výrazem může být libovolná konstanta, atribut překladové gramatiky, nebo operace, jejíž operátory jsou rekurzivně definovány pomocí výrazu. V modelu již není potřeba rozlišovat priority operací, pořadí jejich vyhodnocování určí struktura modelu. Dále není potřeba rozlišovat operaci od relace, neboť relace je chápána jako speciální typ operace nad dvouprvkovou množinou 0, 1. Proto se tato třída souhrnně nazývá RelOp.

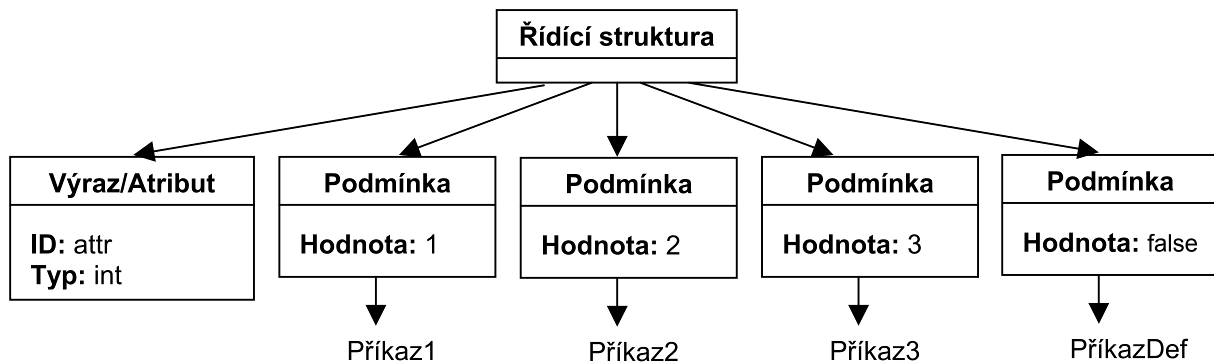
Třída podmínka obsahuje jako atribut konstantu „hodnota“. Tato konstanta může nabývat hodnot buď libovolné ordinální hodnoty nebo speciálně booleovských hodnot true/false. Ke každé podmínce je vázán právě jeden příkaz, který bude vykonán, pokud je vyhodnocený výraz roven dané konstantě. Speciálně pro konstantu true platí: příkaz je vykonán, pokud hodnota výrazu není rovna číslu 0. A speciálně pro konstantu false platí: příkaz je vykonán, pokud nebyl vykonán žádní jiný v rámci aktuální řídící struktury.

Příklad:

Uvažujme následující kód:

```
SWITCH (attr)
   CASE 1:  { Příkaz1 }
   CASE 2:  { Příkaz2 }
   CASE 3:  { Příkaz3 }
   DEFAULT: PříkazDef
```

Část modelu tohoto kódu je následující:



Odpovídající část XML kódu:

```
<SWITCH>
      <EXPRESSION>
            <ATTRIBUTE>
                  <ID> attr </ID>
                  <TYPE> int </TYPE>
            </ATTRIBUTE>
      </EXPRESSION>
      <CONDITIONS>
            <CONDITION>
                  <VALUE> 1 </VALUE>
                  <STATEMENT> …Příkaz1… </STATEMENT>
            </CONDITION>
            <CONDITION>
                  <VALUE> 2 </VALUE>
                  <STATEMENT> …Příkaz2… </STATEMENT>
            </CONDITION>
            <CONDITION>
                  <VALUE> 3 </VALUE>
                  <STATEMENT> …Příkaz3… </STATEMENT>
            </CONDITION>
            <CONDITION>
                  <VALUE> false </VALUE>
                  <STATEMENT> …PříkazDef… </STATEMENT>
            </CONDITION>
      </CONDITIONS>
</SWITCH>
```

Konec příkladu

Příklad:

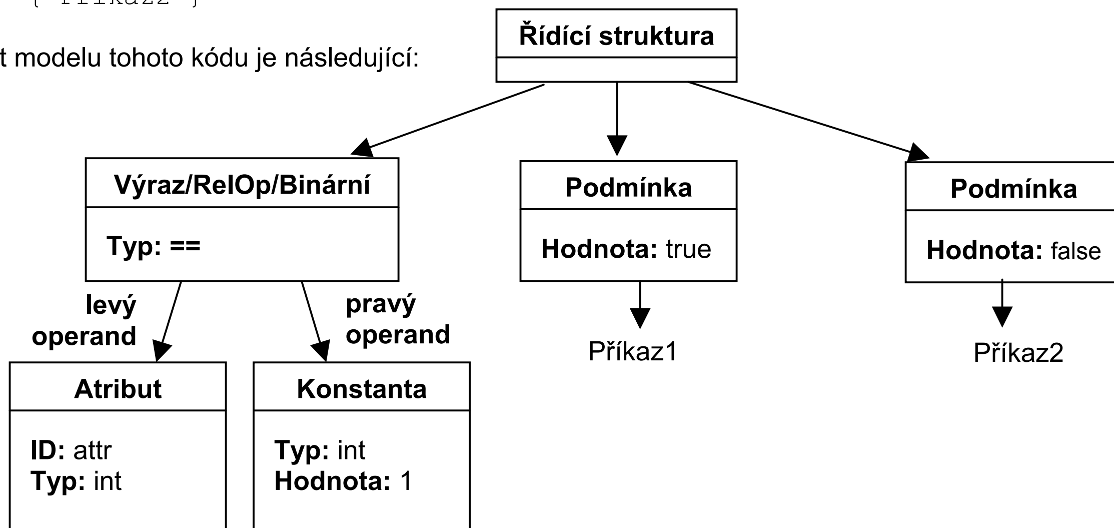
Uvažujme následující kód:

```
SWITCH (attr == 1)
  CASE true:   { Příkaz1 }
  CASE false:  { Příkaz2 }
```

Nebo též ekvivalentní kód:

```
IF (attr == 1)
  { Příkaz1 }
ELSE
  { Příkaz2 }
```

Část modelu tohoto kódu je následující:



Odpovídající část XML kódu:

```
<SWITCH>
      <EXPRESSION>
            <REL_OP>
                  <TYPE> == </TYPE>
                  <LEFT_OP>
                        <ATTRIBUTE>
                              <ID> attr </ID>
                              <TYPE> int </TYPE>
                        </ATTRIBUTE>
                  </LEFT_OP>
                  <RIGHT_OP>
                        <CONSTANT>
                              <TYPE> int </TYPE>
                              <VALUE> 1 </VALUE>
                        </CONSTANT>
                  </RIGHT_OP>
            </REL_OP>
      </EXPRESSION>
      <CONDITIONS>
            <CONDITION>
                  <VALUE> true </VALUE>
                  <STATEMENT> …Příkaz1… </STATEMENT>
            </CONDITION>
            <CONDITION>
                  <VALUE> false </VALUE>
                  <STATEMENT> …Příkaz2… </STATEMENT>
            </CONDITION>
      </CONDITIONS>
</SWITCH>
```

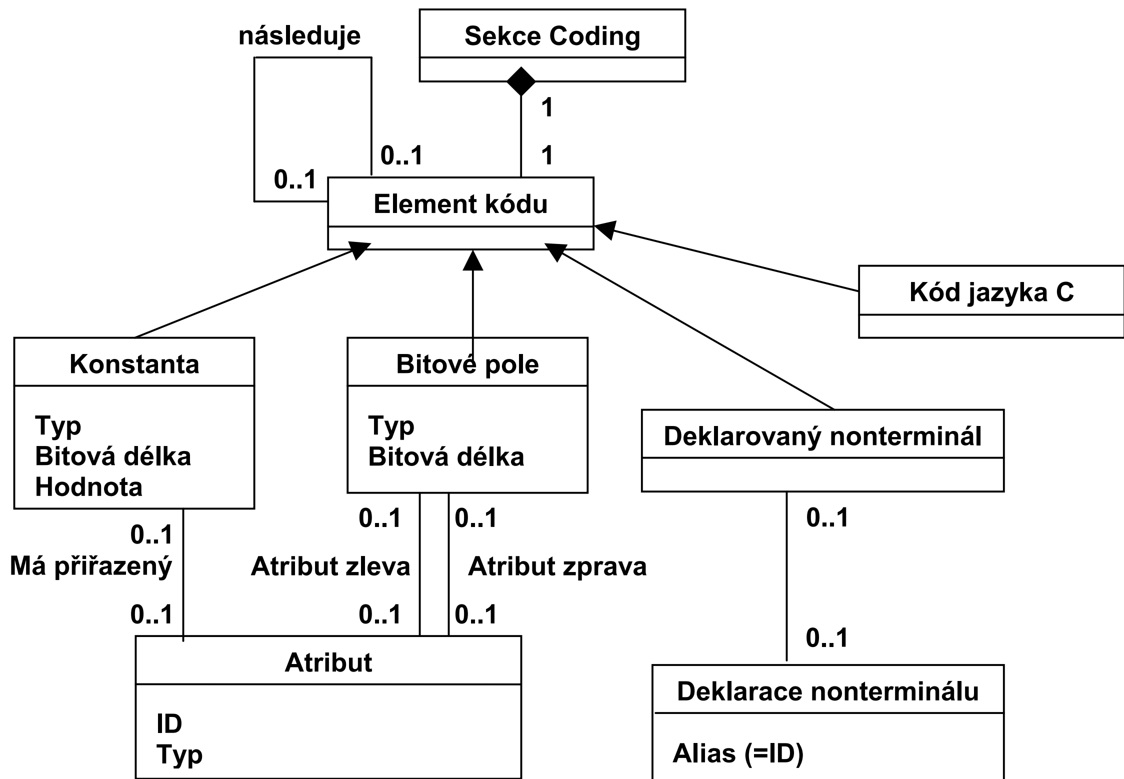
Poznámka: Model pro podmínku if bez sekce else je vytvořen podobně jako v předchozím příkladu pouze s rozdílem, že by neobsahoval podmínku s konstantou false.

2.2.2.2 Atomární příkazy

Atomární příkazy jsou takové příkazy, které obsahují již bez řídící struktury pouze sekci Behavior, Asembler nebo Coding.

2.2.2.2.1 Sekce Coding

Sekce coding slouží k popisu zakódování dané instrukce do binárního kódu. Následující model ukazuje strukturu této sekce:



Sekce coding se skládá z jednotlivých elementů kódu dané operace. Tímto elementem může být konstanta, která je charakterizována bitovou délkou a její hodnotou. Konstanta může být popřípadě přiřazena atributu, aby bylo možné její hodnotu měnit (což se ovšem nedoporučuje).

Dalším elementem je proměnná. Proměnná je bitové pole dané délky, jehož hodnota se koresponduje s daným atributem. Každé proměnné můžou být přiřazeny maximálně 2 atributy, přičemž jeden koresponduje překladu z assemblerovských instrukcí do binárního kódu a druhý atribut koresponduje inverznímu překladu.

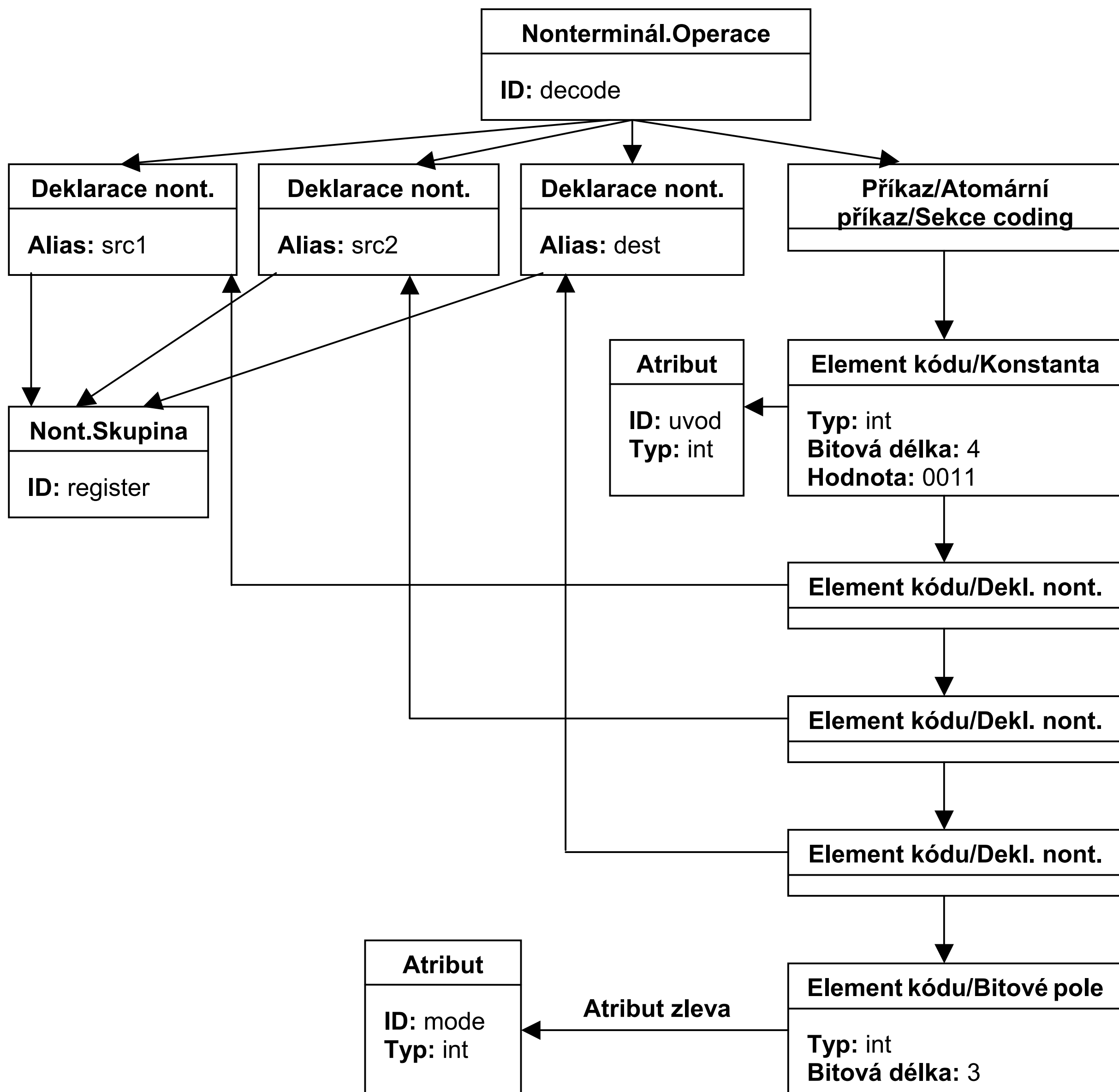
Elementem může být také nonterminál, který již byl pro tuto operaci deklarován. Tímto nonterminálem je buď jistá skupina nebo konkrétní operace. Místo, kde se nachází právě tento nonterminál, bude vyplněno tím binárním kódem, který obsahuje příslušná operace. To může být hierarchicky zanořeno do jisté konečné úrovně.

Příklad:

Uvažujme následující kód:

```
OPERATION decode {
  INSTANCE register ALIAS  { src1, src2, dest } ;
  CODING { uvod=0b0011 src1 src2 dest mode=0bx[3] }
        ...
        }
```

Část modelu tohoto kódu je následující:



Odpovídající část XML kódu:

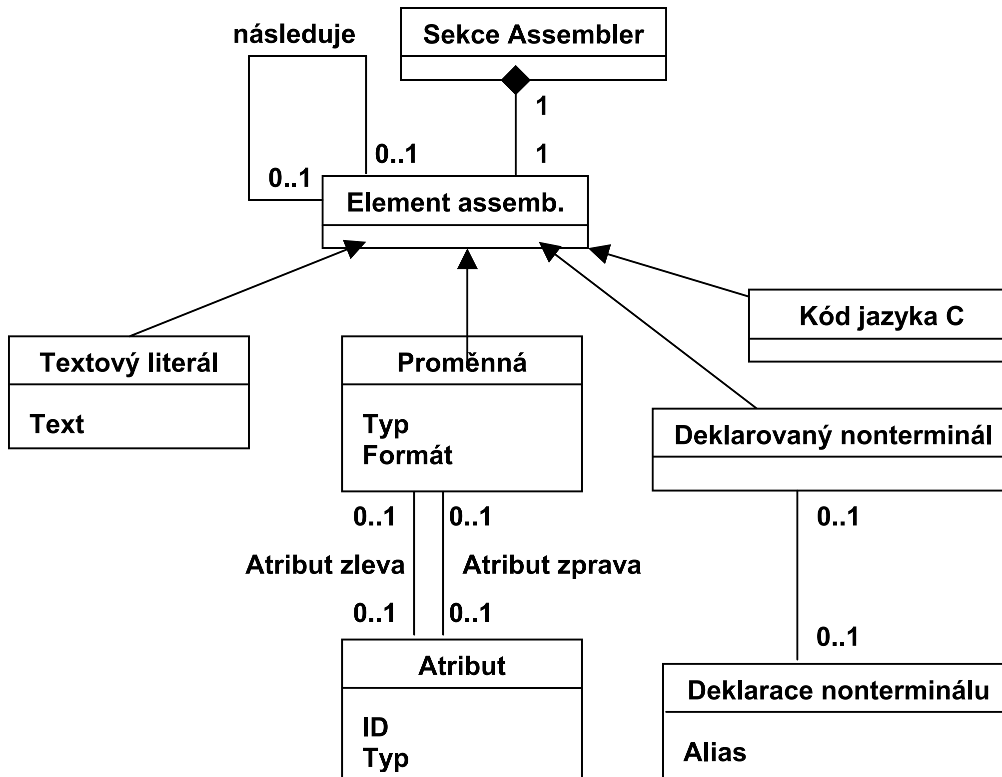
```
<OPERATION>
      <ID> decode </ID>
      <DECLARATIONS>
            <DECLARATION>
                  <NONTERMINAL> register </NONTERMINAL>
                  <ALIAS> scr1 </ALIAS>
            </DECLARATION>
            <DECLARATION>
                  <NONTERMINAL> register </NONTERMINAL>
                  <ALIAS> scr2 </ALIAS>
            </DECLARATION>
            <DECLARATION>
                  <NONTERMINAL> register </NONTERMINAL>
                  <ALIAS> dest </ALIAS>
            </DECLARATION>
      </DECLARATIONS>
```

```
<CODE>
      <CODING>
            <CONSTANT>
                  <TYPE> int </TYPE>
                  <BIT_LENGTH> 4 </BIT_LENGTH>
                  <VALUE> 0011 </VALUE>
                  <LEFT_ATTRIBUTE> uvod </LEFT_ATTRIBUTE>
            </CONSTANT>
            <NONTERMINAL>
                  <ID> src1 </ID>
            </NONTERMINAL>
            <NONTERMINAL>
                  <ID> src2 </ID>
            </NONTERMINAL>
            <NONTERMINAL>
                  <ID> dest </ID>
            </NONTERMINAL>
            <BIT_FIELD>
                  <TYPE> int </TYPE>
                  <BIT_LENGTH> 3 </BIT_LENGTH>
                  <LEFT_ATTRIBUTE> mode </LEFT_ATTRIBUTE>
            </BIT_FIELD >
      </CODING>
</CODE>
```

Konec příkladu

2.2.2.2.2 Sekce Assembler

Sekce assembler slouží k popisu syntaktické struktury dané operace v assemblerovském kódu. Následující model ukazuje strukturu této sekce:



Sekce assembler se skládá z jednotlivých elementů assemblerovských instrukcí. Tímto elementem může být textový literál, který bude testován na schodu.

Dalším elementem je proměnná. Proměnná je jistý atribut z překladové gramatiky, který má navíc specifikovaný svůj typ a hlavně formát. Atribut formát nabývá následujících hodnot:

- #U (**hodnota bez znaménka**)
- #X (**šestnáctková číslice**)
- #S *konstantní numerický výraz* (**hodnota se znaménkem** s určením pozice znaménkového bitu)

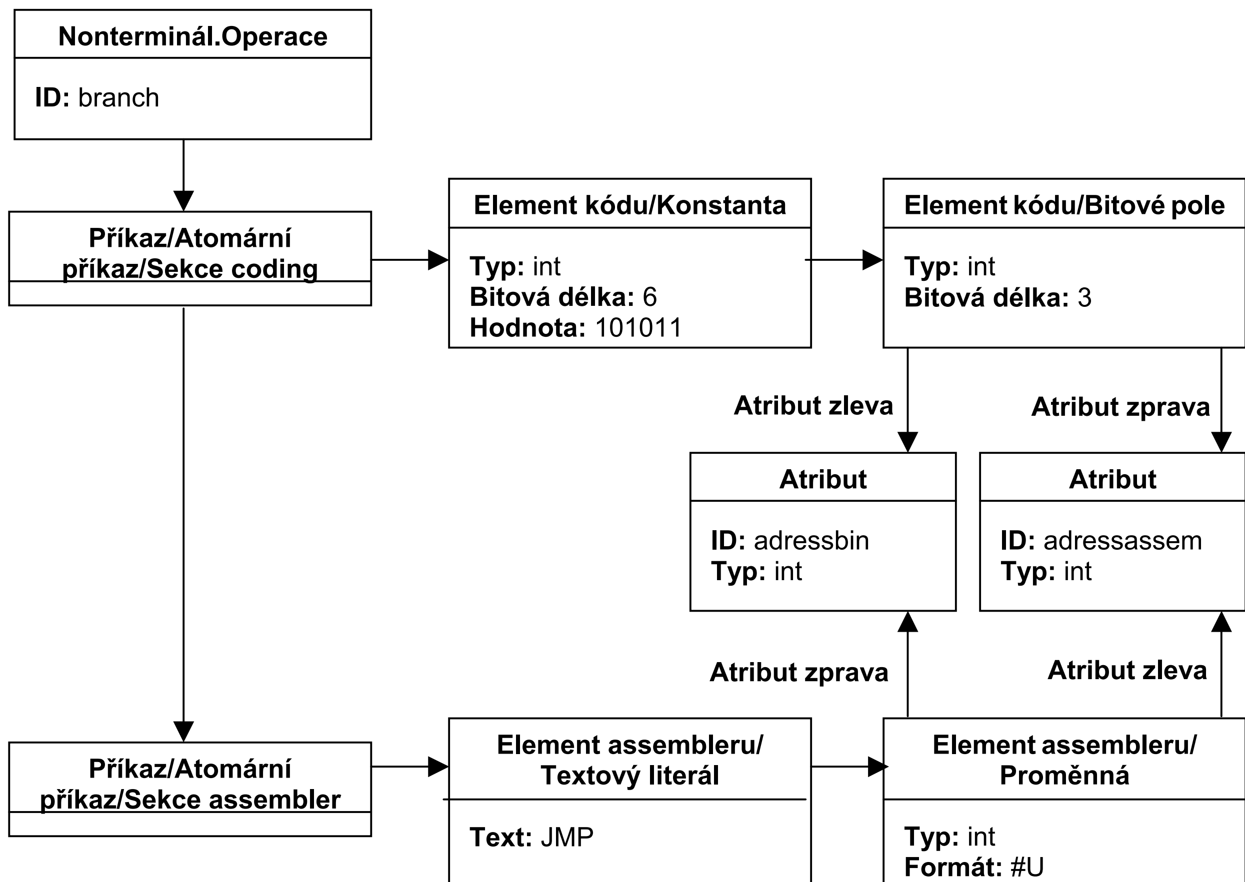
Elementem může být také nonterminál, který již byl pro tuto operaci deklarován. Tímto nonterminálem je buď jistá skupina nebo konkrétní operace. Místo, kde se nachází právě tento nonterminál, bude vyplněno těmi asseblerovskými instrukcemi, které obsahuje příslušná operace. To může být hierarchicky zanořeno do jisté konečné úrovně.

Příklad:

Uvažujme následující kód:

```
OPERATION branch {
  CODING { 0b101011 addressbin=0bx[10]=addressassem }
  ASSEMBLER { "JMP" addressassem=#U=addressbin }
  }
```

Část modelu tohoto kódu je následující:



Odpovídající část XML kódu:

```
<OPERATION>
      <CODE>
            <CODING>
                  <CONSTANT>
                        <TYPE> int </TYPE>
                        <BIT_LENGTH> 6 </BIT_LENGTH>
                        <VALUE> 101011 </VALUE>
                  </CONSTANT>
                  <BIT_FIELD>
                        <TYPE> int </TYPE>
                        <BIT_LENGTH> 3 </BIT_LENGTH>
                        <LEFT_ATTRIBUTE> adressbin </LEFT_ATTRIBUTE>
                        <RIGHT_ATTRIBUTE> adressassem </RIGHT_ATTRIBUTE>
                  </BIT_FIELD >
            <CODING>
            <ASSEMBLER>
                  <TEXT> JMP </TEXT>
                  <VARIABLE>
                        <TYPE> int </TYPE>
                        <FORMAT> #U </FORMAT>
                  </VARIABLE>
            </ASSEMBLER>
      </CODE>
</OPERATION>
```

Konec příkladu

2.2.2.2.3 Sekce Behavior

Sekce behavior slouží k popisu chování dané instrukce. Následující model ukazuje strukturu této sekce: