Università di Bergamo

Relazione progetto

Linguaggi Formali e Compilatori

Parser ISA Risc-V

Autore:
Dario Sardi

Supervisore:
Giuseppe Psaila

12 Marzo 2019

Abstract

Il progetto è volto allo sviluppo di un traduttore di un subset di istruzioni appartenenti all'ISA RISC-V da linguaggio assembly a linguaggio naturale.

1 Output e Input

Il programma prende come input il testo contenuto nel file "resources/input.file" e restituisce la traduzione in linguaggio naturale come output su terminale seguito da una eventuale lista di errori.
Un esempio di output privo di errori:

```
1     ----- PARSING STARTED-------
2 created ADDI between 1 and 12 to be saved at 10
3 created OR between 12 and 2 to be saved at 5
4 added label li
5 created ADD between 7 and 5 to be saved at 22
6 created AND between 7 and 5 to be saved at 23
7 jump to li at line 3 if condition is true
8 created OR between 22 and 23 to be saved at 24
9 ----- PARSING DONE -------
```

In caso di errori l'output è il seguente:

```
1     ----- PARSING STARTED-------
2 created ADDI between 1 and 12 to be saved at 10
3 created OR between 12 and 2 to be saved at 5
4 added label li
5 created ADD between 7 and 5 to be saved at 22
6 created AND between 7 and 45 to be saved at 23
7
8 ##############################
9 #  ERROR! PARSING STOPPED!  #
10 ##############################
11 ERRORS:
12 1. label 'li' at line 4 already exist, ignoring it.
13 2. Lexer error at [6:15].Max register value is 30!
```

2 Registri e immediati

Le specifiche scelte per questo progetto prevedono 30 registri indirizzabili attraverso la sequenza '0xNumeroDiRegistro' dove il numero del registro è un numero intero compreso fra 0 e 30.
La scelta del prefisso è stata dettata dalla necessità di distinguere numeri interi e numeri dei registri durante il parsing e rendere il meno confusionario possibile il codice.
Per quanto riguarda gli immediati è stata prevista la dimensione massima di una word su ISA RISC-V equivalente al valore decimale 4096.

3 Variabili

È previsto l'utilizzo di variabili statiche per etichettare registri o valori notevoli. La sintassi per dichiarare il nome di un registro è:

```
DR etichetta 0xNumeroDiRegistro
```

Una definizione valida può essere dunque:

```
DR RegistroA 0x9
```

Si ricorda che il valore massimo assegnabile ad un registro è pari a 0x30 per specifiche precedentemente definite. Per quanto riguarda le variabili, si possono creare variabili numeriche statiche di tipo Byte e Word. Risulta possibile riservare una variabile senza inizializzarla attraverso il comando RESB per riservare un Byte e RESW per riservare una Word.
Esempio di variabili riservate sono:

```
RESW valoreMassimo
RESB threshold
```

Questa operazione è stata introdotta per una implementazione futura dove variabili dinamiche potranno essere assegnate o variate utilizzandole come destinazione all'interno di funzioni. Il comando attualmente utile risulta essere la dichiarazione di una variabile con relativa inizializzazione effettuabile tramite:

```
DW valoreMassimo 3623
```

```
    DB threshold 90
```

3.1 Error handling

Nel caso di una dichiarazione di variabile troppo grande il programma risponderà settando la variabile a 0 e proseguendo normalmente.
Esempi di input-output possibili sono:

1: input

```
1   DW ciao 5400
```

2: output

```
1   ----- PARSING STARTED-------
2    defined new Word-type variable with value 0
3    named ciao
4   ----- PARSING DONE -------
5   ERRORS:
6   1.   variable value is not Word type
7        (max value 4096), ciao set to 0
```

Oppure:

3: input

```
1   DB ciao 300
```

4: output

```
1   ----- PARSING STARTED-------
2    defined new Byte-type variable with value 0
3    named ciao
4   ----- PARSING DONE -------
5   ERRORS:
6   1.   variable value is not Byte type
7        (max value 256), ciao set to 0
```

Altro errore può verificarsi nella dichiarazione di un registro dal numero superiore al 30 come ad esempio:

5: input

```
1    DR reg1 0x50
```

6: output

```
1    ----- PARSING STARTED-------
2    variable reg too big for register-type, set to 0
3    ----- PARSING DONE -------
4    ERRORS:
5    1.  Lexer error at [1:9].Max register
6        value is 30!
```

Nel caso di un assegnamento ad una variabile registro un valore immediato, l'errore viene considerato grave e il parsing interrotto.

4 Le funzioni

Il subset di istruzioni considerato prevede due tipologie di funzioni, le R-type e le I-type.

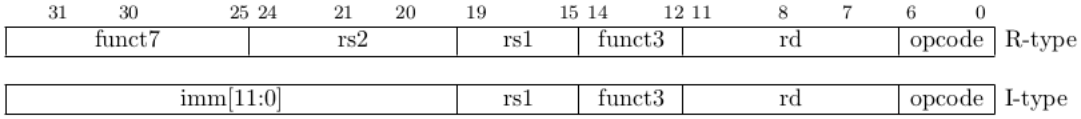

Figure 1: Struttura in binario delle funzioni e relativi attributi

In entrambe le funzioni la struttura sintattica risulta essere

```
Funzione Destinazione Operando1 Operando2
```

Per le funzioni R-type destinazione e operandi saranno registri

```
Funzione RegDestinazione RegOperando1 RegOperando2
```

Per le funzioni di I-type il secondo operando dovrà essere un immediato

```
Funzione RegDestinazione RegOperando1 ImmOperando2
```

Le funzioni previste sono le seguenti:

1. R-Type: ADD,SUB,MUL,OR,XOR,AND
2. I-Type: ADDI,SUBI,ORI,XORI,ANDI

Esempi di funzioni sono :

7: input

```
1 ADDI 0x10 0x01 12
2 OR 0x05 0x12 0x02
```

8: output

```
1 ----- PARSING STARTED-------
2 created ADDI between 1 and 12 to be saved at 10
3 created OR between 12 and 2 to be saved at 5
4 ----- PARSING DONE-------
```

4.1 Error Handling

Gli unici errori possibili in una funzione sono l'utilizzo di numeri di registri errati,di variabili errate (registro al posto di immediato e viceversa) o immediati di dimensione superiore alla word.
Nel caso di registri troppo grandi l'errore è trasmesso direttamente dal controllo su ogni singolo valore di registro e causa l'interruzione del parsing.

9: input

```
1    DR reg 0x10
2    ADD reg reg 0x50
```

10: output

```
1    ----- PARSING STARTED-------
2    defined new Register variable with value
3     10 named reg
4
5    ##############################
6    #  ERROR! PARSING STOPPED!  #
7    ##############################
8    ERRORS:
9    1.  Lexer error at [2:14].Max register value
10       is 30!
```

L'utilizzo di variabili errate è contemplato e causa l'interruzione del parsing:

11: input

```
1    DR reg 0x10
2    DW c 12
3    ADD reg reg c
```

12: output

```
1    ----- PARSING STARTED-------
2    defined new Register variable with value
3     10 named reg
4    defined new Word-type variable with value
5     12 named c
6
7    ##############################
8    #  ERROR! PARSING STOPPED!  #
9    ##############################
10   ERRORS:
11   1.  at line 3 using a non register variable
```

5 Jump

Sono state implementate chiamate di salto condizionate e non, non prevedendo chiamate a label non ancora definiti (jump ahead). Le jump consistono in un label definito come una stringa seguita da due punti e una chiamata di salto al label stesso.
Chiamate incondizionate si effettuano tramite "JMP" seguito dal label desiderato:

13: input

```
1    jumplabel :
2    DR reg 0x10
3    ADD reg reg 0x20
4    JMP jumplabel
```

14: output

```
1    ----- PARSING STARTED-------
2    added label jumplabel
3    defined new Register variable with value 10
4     named reg
5    created ADD between 10 and 20 to be saved at 10
6    jump to jumplabel at line 1
7    ----- PARSING DONE -------
```

Per salti condizionati si ha una comparazione fra valori di registro che nel caso risulti corretta permette il salto.
Funzioni di comparazione possono essere:

1. JE/JZ Jump Equal or Jump Zero
2. JNE/JNZ Jump not Equal or Jump Not Zero
3. JG/JNLE Jump Greater or Jump Not Less/Equal
4. JGE Jump Greater/Equal
5. JL/JNGE Jump Less or Jump Not Greater/Equal
6. JLE/JNG Jump Less/Equal or Jump Not Greater

Esempio di salto condizionato:

15: input

```
1    jumplabel :
2    DR reg 0x10
3    ADD reg reg 0x20
4    CMP 0x10 0x11
5    JNE jumplabel
```

16: output

```
1    ----- PARSING STARTED-------
2    added label jumplabel
3    defined new Register variable with value 10
4     named reg
5    created ADD between 10 and 20 to be saved at 10
6    comparing register 10 to 11
7    jump to jumplabel at line 1
8     if condition is true
9    ----- PARSING DONE -------
```

5.1 Error Handling

Nel caso di label ripetuti solo il primo viene considerato e i successivi vengono ignorati.

17: input

```
1    jumplabel :
2    DR reg 0x10
3    ADD reg reg 0x20
4    jumplabel :
5    CMP 0x10 0x11
6    JNE jumplabel
```

18: output

```
1    ----- PARSING STARTED-------
2    added label jumplabel
3    defined new Register variable with value 10
4     named reg
5    created ADD between 10 and 20 to be saved at 10
6    comparing register 10 to 11
7    jump to jumplabel at line 1
8     if condition is true
9    ----- PARSING DONE -------
10   ERRORS:
11   1.  label 'jumplabel' at line 4 already exist,
12       ignoring it.
```

Nel caso di salti a label non esistenti il programma si interrompe segnalando l'errore.

19: input

```
1    jumplabel :
2    DR reg 0x10
3    ADD reg reg 0x20
4    CMP 0x10 0x11
5    JNE somewhere
```

20: output

```
1    ----- PARSING STARTED-------
2    added label jumplabel
3    defined new Register variable with value 10
4     named reg
5    created ADD between 10 and 20 to be saved at 10
6    comparing register 10 to 11
7
8    ##############################
9    #  ERROR! PARSING STOPPED!  #
10   ##############################
11   ERRORS:
12   1.  Label somewhere at line 5 does not exist!
13       STOPPING PARSER!
```

6 ANTLR grammatica

Variabili dichiarate per il lexer:

21: lexer

```
1
2  TWODOT  :       ':';
3  CMP     :       'CMP';
4  DTYPE   :       'DB'|'DW';
5  VTYPE   :       'RESB' | 'RESW';
6  RTYPE   :       'DR'| 'DRR';
7  JMP     :       'JMP' | 'JE'  | 'JZ' |
8                  'JNE' | 'JNZ' | 'JG' |
9                  'JNLE'| 'JGE' | 'JNG'|
10                 'JL'  |'JNGE' | 'JLE';
11 ADDI    :       'addi'  | 'ADDI';
12 SUBI    :       'subi'  | 'SUBI';
13 ANDI    :       'andi'  | 'ANDI';
14 ORI     :       'ori'   | 'ORI;';
15 XORI    :       'xori'  | 'XORI';
16 ADD     :       'add'   | 'ADD';
17 SUB     :       'sub'   | 'SUB';
18 MUL     :       'mul'   | 'MUL';
19 XOR     :       'xor'   | 'XOR';
20 OR      :       'or'    | 'OR';
21 AND     :       'and'   | 'AND';
22 INT     :       ('0'..'9')+;
23 WS      :   (
24         ' '
25         |'\t'
26             | '\r'
27             )+ {$channel=HIDDEN;}
28         ;
29
30 STRING  :       ('a'..'z'|'A'..'Z')+;
31 ERROR : . {System.out.println("what?...");} ;
```

Il parser inizia considerando come istruzioni valide funzioni, dichiarazioni di variabili, label e jump separate dal terminatore di linea. La grammatica priva delle parti relative a java si presenta cosi:

22: parser

```
1
2   start           : line* ;
3
4   line            : (r3Type | r3IType | defineVar | reserveVar |
5                      defineRegister | jumpUnc | jumpCond | label
6                      | ERROR ) '\n';
7
8   defineVar       : DTYPE STRING INT ;
9
10  reserveVar      : VTYPE STRING ;
11
12  defineRegister  : RTYPE STRING register ;
13
14  r3IType         : fI register register ( immediateVar | INT );
15
16  r3Type          : f3 register register register ;
17
18  f3              : ADD | SUB | MUL | XOR | OR | AND ;
19
20  fI              : ADDI | ANDI | ORI | SUBI | XORI ;
21
22  register        : '0x' INT  | STRING ;
23
24  immediateVar    : STRING ;
25
26  jumpCond        : condition '\n' jumpUnc  ;
27
28  condition       : CMP register register ;
29
30  jumpUnc         : JMP STRING ;
31
32  label           : STRING TWODOT ;
```