コンピュータアーキテクチャ　レポート

03-160441 土屋潤一郎

1. アセンブラの Perl での実装

省略。『実践 コンピュータアーキテクチャ』　第 6 章のソースコードを用いた。

2. 簡単なアセンブルの実行とその結果

1+1=2 を行うため、以下のようなアセンブラを記述した。

```
addi r1, r0, 1
addi r2, r1, 1
end: j end
```

このアセンブラの機械語への翻訳結果は、以下の通りである。

```
000001_00000_00001_0000000000000001_
000001_00001_00010_0000000000000001_
101000_00000000000000000000000010_
```

3. bgt0_sub の追加

3.1 命令の仕様

動作は以下のように定めた。

rt <- rt - rs
if(rt > 0) PC <- PC + dpl else PC <- PC + 1

I 型命令であるから、機械語は以下のようになっている。

これを踏まえて、OP は 36、ニューメリックを bgs とし、アセンブリ言語による表現は、

```
bgs [rs] [rt] [jump 先]
```

とした。

3.2 アセンブラの改良

Perl のソースのうち機械語の出力を行う if 文に、次の 1 行を加える。

```
elsif ($op eq "bgs"){p_b(6,36); p_r2b($f2, $f3); p_b(16, $labels{$f4}-$i-1);print("\n");}
```

3.3 プロセッサの設計変更

教科書第 7 章の設計を変更していく。まず、実行（execute）モジュールの、alu に行わせる計算の種類を指定する opr_gen 関数中の case 文に、以下の 1 行を追加する。

```
6'd36:opr_gen = 5'd20; //OP=36 なら alu に 20 番の計算を行わせる
```

その上で、alu の中身に、20 番の計算として次の 1 行を追加する。

```
5'd20: alu = operand2 - operand1; //制御信号 20 番は、2 番とは逆の減算
```

5'd2: alu = operand1 - operand2; の計算を用いないのは、変数の並びをジャンプ系の命令にあわせてしまったからである。

次に、計算結果を生成する回路の case の中の 1 文目を書き換える。

```
6'd0, 6'd1, 6'd4, 6'd5, 6'd6, 6'd36: calc = alu_result; //6'd36（OP）を追加
```

さらに、次のプレイヤカウンタの値を、素直に次の命令にするか、ジャンプするのかを決める回路の中を決める回路にも、OP が 36 の時の動作を書き加える。

```
6'd34, 6'd36: npc = (reg1 < reg2)? branch : nonbranch; //6'd36（OP）を追加
```

これは blt と同じ条件である。

結果を書き込むレジスタを決める回路にも、OP が 36 の時の動作を書き加えてやる。

```
6'd1, 6'd3, 6'd4, 6'd5, 6'd6, 6'd16, 6'd18, 6'd20, 6'd36: wreg = rt;
//6'd36 でも結果を rt に書き込む
```

最後に、実行モジュールの assign 文のうち、alu の operand2 に入る変数を決める文を書き換える。

```
assign operand2 = (op == 6'd0 || op == 6'd36) ? reg2: dpl_imm;
//operand2 はＩ型算術演算命令と同じ
```

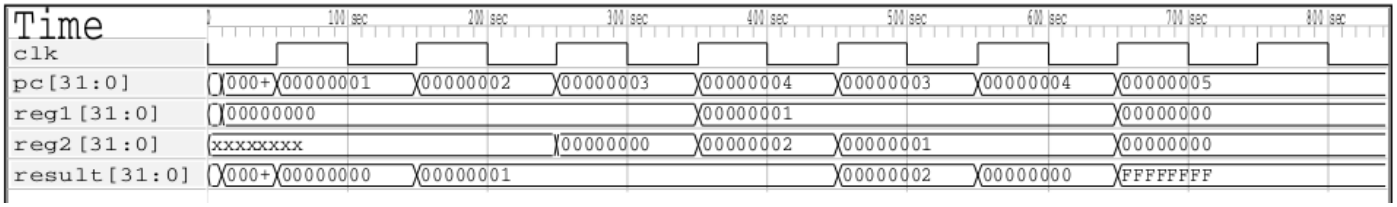
これで改変箇所は終わりである。

3.4 bgs の動作確認

動作を確認するために、以下のようなアセンブラを書いた。

```
addi r1 r0 2
addi r2 r0 0
addi r3 r0 1
label: addi r2 r2 1
bgs r3 r1 label
end: j end
```

これなら、pc=3 と pc=4 が 2 回実行されるはずである。この結果は以下の通り。



PC を見ると、3 行目と 4 行目を 2 回繰り返していることがわかる。

3.5 新命令を用いない 1~20 の階和計算

アセンブリ言語は以下の通り。

```
addi r1, r0, 20
addi r2, r0, 0
addi r3, r0, 0
label:  addi r2,r2,1
add r3, r2, r3
blt r2, r1, label
add r3, r0, r3
end: j end
```

これは、gtkwave による観察を掲載するのは困難なので、代わりに$monitor を用いた観測を以って動作の確認とする。

```
   0 PC:  x reg1: 0xxxxxxxxx reg2: 0xxxxxxxxx result: 0xffffffff
  10 PC:  0 reg1: 0x00000000 reg2: 0xxxxxxxxx result: 0x00000014
  50 PC:  1 reg1: 0x00000000 reg2: 0xxxxxxxxx result: 0x00000000
 150 PC:  2 reg1: 0x00000000 reg2: 0xxxxxxxxx result: 0x00000000
 250 PC:  3 reg1: 0x00000000 reg2: 0x00000000 result: 0x00000001
 350 PC:  4 reg1: 0x00000001 reg2: 0x00000000 result: 0x00000001
 450 PC:  5 reg1: 0x00000001 reg2: 0x00000014 result: 0xffffffff
 550 PC:  3 reg1: 0x00000001 reg2: 0x00000001 result: 0x00000002
 650 PC:  4 reg1: 0x00000002 reg2: 0x00000001 result: 0x00000003
 750 PC:  5 reg1: 0x00000002 reg2: 0x00000014 result: 0xffffffff
 850 PC:  3 reg1: 0x00000002 reg2: 0x00000002 result: 0x00000003
 950 PC:  4 reg1: 0x00000003 reg2: 0x00000003 result: 0x00000006
1050 PC:  5 reg1: 0x00000003 reg2: 0x00000014 result: 0xffffffff
1150 PC:  3 reg1: 0x00000003 reg2: 0x00000003 result: 0x00000004
1250 PC:  4 reg1: 0x00000004 reg2: 0x00000006 result: 0x0000000a
1350 PC:  5 reg1: 0x00000004 reg2: 0x00000014 result: 0xffffffff
1450 PC:  3 reg1: 0x00000004 reg2: 0x00000004 result: 0x00000005
1550 PC:  4 reg1: 0x00000005 reg2: 0x0000000a result: 0x0000000f
1650 PC:  5 reg1: 0x00000005 reg2: 0x00000014 result: 0xffffffff
1750 PC:  3 reg1: 0x00000005 reg2: 0x00000005 result: 0x00000006
1850 PC:  4 reg1: 0x00000006 reg2: 0x0000000f result: 0x00000015
1950 PC:  5 reg1: 0x00000006 reg2: 0x00000014 result: 0xffffffff
2050 PC:  3 reg1: 0x00000006 reg2: 0x00000006 result: 0x00000007
2150 PC:  4 reg1: 0x00000007 reg2: 0x00000015 result: 0x0000001c
2250 PC:  5 reg1: 0x00000007 reg2: 0x00000014 result: 0xffffffff
2350 PC:  3 reg1: 0x00000007 reg2: 0x00000007 result: 0x00000008
2450 PC:  4 reg1: 0x00000008 reg2: 0x0000001c result: 0x00000024
```

```
2550 PC:  5 reg1: 0x00000008 reg2: 0x00000014 result: 0xffffffff
2650 PC:  3 reg1: 0x00000008 reg2: 0x00000008 result: 0x00000009
2750 PC:  4 reg1: 0x00000009 reg2: 0x00000024 result: 0x0000002d
2850 PC:  5 reg1: 0x00000009 reg2: 0x00000014 result: 0xffffffff
2950 PC:  3 reg1: 0x00000009 reg2: 0x00000009 result: 0x0000000a
3050 PC:  4 reg1: 0x0000000a reg2: 0x0000002d result: 0x00000037
3150 PC:  5 reg1: 0x0000000a reg2: 0x00000014 result: 0xffffffff
3250 PC:  3 reg1: 0x0000000a reg2: 0x0000000a result: 0x0000000b
3350 PC:  4 reg1: 0x0000000b reg2: 0x00000037 result: 0x00000042
3450 PC:  5 reg1: 0x0000000b reg2: 0x00000014 result: 0xffffffff
3550 PC:  3 reg1: 0x0000000b reg2: 0x0000000b result: 0x0000000c
3650 PC:  4 reg1: 0x0000000c reg2: 0x00000042 result: 0x0000004e
3750 PC:  5 reg1: 0x0000000c reg2: 0x00000014 result: 0xffffffff
3850 PC:  3 reg1: 0x0000000c reg2: 0x0000000c result: 0x0000000d
3950 PC:  4 reg1: 0x0000000d reg2: 0x0000004e result: 0x0000005b
4050 PC:  5 reg1: 0x0000000d reg2: 0x00000014 result: 0xffffffff
4150 PC:  3 reg1: 0x0000000d reg2: 0x0000000d result: 0x0000000e
4250 PC:  4 reg1: 0x0000000e reg2: 0x0000005b result: 0x00000069
4350 PC:  5 reg1: 0x0000000e reg2: 0x00000014 result: 0xffffffff
4450 PC:  3 reg1: 0x0000000e reg2: 0x0000000e result: 0x0000000f
4550 PC:  4 reg1: 0x0000000f reg2: 0x00000069 result: 0x00000078
4650 PC:  5 reg1: 0x0000000f reg2: 0x00000014 result: 0xffffffff
4750 PC:  3 reg1: 0x0000000f reg2: 0x0000000f result: 0x00000010
4850 PC:  4 reg1: 0x00000010 reg2: 0x00000078 result: 0x00000088
4950 PC:  5 reg1: 0x00000010 reg2: 0x00000014 result: 0xffffffff
5050 PC:  3 reg1: 0x00000010 reg2: 0x00000010 result: 0x00000011
5150 PC:  4 reg1: 0x00000011 reg2: 0x00000088 result: 0x00000099
5250 PC:  5 reg1: 0x00000011 reg2: 0x00000014 result: 0xffffffff
5350 PC:  3 reg1: 0x00000011 reg2: 0x00000011 result: 0x00000012
5450 PC:  4 reg1: 0x00000012 reg2: 0x00000099 result: 0x000000ab
5550 PC:  5 reg1: 0x00000012 reg2: 0x00000014 result: 0xffffffff
5650 PC:  3 reg1: 0x00000012 reg2: 0x00000012 result: 0x00000013
5750 PC:  4 reg1: 0x00000013 reg2: 0x000000ab result: 0x000000be
5850 PC:  5 reg1: 0x00000013 reg2: 0x00000014 result: 0xffffffff
5950 PC:  3 reg1: 0x00000013 reg2: 0x00000013 result: 0x00000014
6050 PC:  4 reg1: 0x00000014 reg2: 0x000000be result: 0x000000d2
6150 PC:  5 reg1: 0x00000014 reg2: 0x00000014 result: 0xffffffff
6250 PC:  6 reg1: 0x00000000 reg2: 0x00000000 result: 0xffffffff
```

6050 秒の result に注目すると、1~20 までの階和である 210 の 16 進法表記である 0xd2 が確認できる。

3.6 新命令を用いた 1~20 の階和計算

アセンブリ言語は以下の通り。

```
addi r1, r0, 20
addi r2, r0, 1
addi r3, r0, 0
label:  add r3, r1, r3
bgs r2, r1, label
end: j end
```

こちらも$monitor による出力結果を示す。

```
   0 PC:  x reg1: 0xxxxxxxxx reg2: 0xxxxxxxxx result: 0xffffffff
  10 PC:  0 reg1: 0x00000000 reg2: 0xxxxxxxxx result: 0x00000014
  50 PC:  1 reg1: 0x00000000 reg2: 0xxxxxxxxx result: 0x00000001
 150 PC:  2 reg1: 0x00000000 reg2: 0xxxxxxxxx result: 0x00000000
 250 PC:  3 reg1: 0x00000014 reg2: 0x00000000 result: 0x00000014
 350 PC:  4 reg1: 0x00000001 reg2: 0x00000014 result: 0x00000013
 450 PC:  3 reg1: 0x00000013 reg2: 0x00000014 result: 0x00000027
 550 PC:  4 reg1: 0x00000001 reg2: 0x00000013 result: 0x00000012
 650 PC:  3 reg1: 0x00000012 reg2: 0x00000027 result: 0x00000039
 750 PC:  4 reg1: 0x00000001 reg2: 0x00000012 result: 0x00000011
 850 PC:  3 reg1: 0x00000011 reg2: 0x00000039 result: 0x0000004a
 950 PC:  4 reg1: 0x00000001 reg2: 0x00000011 result: 0x00000010
1050 PC:  3 reg1: 0x00000010 reg2: 0x0000004a result: 0x0000005a
1150 PC:  4 reg1: 0x00000001 reg2: 0x00000010 result: 0x0000000f
1250 PC:  3 reg1: 0x0000000f reg2: 0x0000005a result: 0x00000069
1350 PC:  4 reg1: 0x00000001 reg2: 0x0000000f result: 0x0000000e
1450 PC:  3 reg1: 0x0000000e reg2: 0x00000069 result: 0x00000077
1550 PC:  4 reg1: 0x00000001 reg2: 0x0000000e result: 0x0000000d
1650 PC:  3 reg1: 0x0000000d reg2: 0x00000077 result: 0x00000084
1750 PC:  4 reg1: 0x00000001 reg2: 0x0000000d result: 0x0000000c
1850 PC:  3 reg1: 0x0000000c reg2: 0x00000084 result: 0x00000090
1950 PC:  4 reg1: 0x00000001 reg2: 0x0000000c result: 0x0000000b
2050 PC:  3 reg1: 0x0000000b reg2: 0x00000090 result: 0x0000009b
2150 PC:  4 reg1: 0x00000001 reg2: 0x0000000b result: 0x0000000a
2250 PC:  3 reg1: 0x0000000a reg2: 0x0000009b result: 0x000000a5
2350 PC:  4 reg1: 0x00000001 reg2: 0x0000000a result: 0x00000009
2450 PC:  3 reg1: 0x00000009 reg2: 0x000000a5 result: 0x000000ae
2550 PC:  4 reg1: 0x00000001 reg2: 0x00000009 result: 0x00000008
2650 PC:  3 reg1: 0x00000008 reg2: 0x000000ae result: 0x000000b6
2750 PC:  4 reg1: 0x00000001 reg2: 0x00000008 result: 0x00000007
2850 PC:  3 reg1: 0x00000007 reg2: 0x000000b6 result: 0x000000bd
2950 PC:  4 reg1: 0x00000001 reg2: 0x00000007 result: 0x00000006
3050 PC:  3 reg1: 0x00000006 reg2: 0x000000bd result: 0x000000c3
3150 PC:  4 reg1: 0x00000001 reg2: 0x00000006 result: 0x00000005
3250 PC:  3 reg1: 0x00000005 reg2: 0x000000c3 result: 0x000000c8
3350 PC:  4 reg1: 0x00000001 reg2: 0x00000005 result: 0x00000004
3450 PC:  3 reg1: 0x00000004 reg2: 0x000000c8 result: 0x000000cc
3550 PC:  4 reg1: 0x00000001 reg2: 0x00000004 result: 0x00000003
3650 PC:  3 reg1: 0x00000003 reg2: 0x000000cc result: 0x000000cf
3750 PC:  4 reg1: 0x00000001 reg2: 0x00000003 result: 0x00000002
3850 PC:  3 reg1: 0x00000002 reg2: 0x000000cf result: 0x000000d1
3950 PC:  4 reg1: 0x00000001 reg2: 0x00000002 result: 0x00000001
4050 PC:  3 reg1: 0x00000001 reg2: 0x000000d1 result: 0x000000d2
4150 PC:  4 reg1: 0x00000001 reg2: 0x00000001 result: 0x00000000
4250 PC:  5 reg1: 0x00000000 reg2: 0x00000000 result: 0xffffffff
```

こちらも 4050 秒に注目すれば、0xd2 が計算結果として出てきているのがわかる。

また、新たな命令を用いることで、計算終了が 2000 秒早くなった。1 クロック 100 秒なので、20 クロック分、性能が向上した。

3.7 性能向上

N=20 のとき 20 クロック速くなったので、性能向上は N である。実際、アセンブラを見ると、繰り返し計算される部分の行数が一行減っている。