第６回 ハードウェア記述言語
～フラグと条件分岐～

中野 秀洋

フラグと条件分岐

命令	ニモニック	操作コード	動作	例
加算	ADD GRa, GRb	2 (0010)	GRa ← GRa + GRb FR ← {CF, ZF}	y += x;
減算	SUB GRa, GRb	3 (0011)	GRa ← GRa - GRb FR ← {CF, ZF}	y -= x;
比較	CMP GRa, GRb	7 (0111)	GRa - GRb FR ← {CF, ZF}	y - x
ゼロジャンプ	JZ ADRS	8 (1000)	if(FR[0] == 1) PR ← ADRS	while (y == x);
キャリージャンプ	JC ADRS	9 (1001)	if(FR[1] == 1) PR ← ADRS	while (y < x);

0111	1111	1000	1000	0000
+ 0001	+ 0001	- 0001	- 1000	- 0001
01000	10000	00111	00000	11111
{0,0}	{1,1}	{0,0}	{0,1}	{1,0}

※ CF: キャリーフラグ（演算結果でキャリー）

※ ZF: ゼロフラグ（演算結果がゼロ）

FR :フラグレジスタ

条件分岐のプログラム(1)

```
if ( n == 10 ) {
    処理1
}
処理2
```

```
    LDI  N
    LD   GR1, GR0
    LDI  10
    CMP  GR1, GR0
    JZ   L1
    JMP  L2
L1: 処理1
L2: 処理2
```

```
if ( n != 10 )
```

```
    ...
    CMP  GR1, GR0
    JZ   L1
    処理1
L1: 処理2
```

```
if ( n < 10 ) {
    処理1
}
処理2
```

```
    LDI  N
    LD   GR1, GR0
    LDI  10
    CMP  GR1, GR0
    JC   L1
    JMP  L2
L1: 処理1
L2: 処理2
```

```
if ( n >= 10 )
```

```
    ...
    CMP  GR1, GR0
    JC   L1
    処理1
L1: 処理2
```

```
if ( n > 10 ) {
    処理1
}
処理2
```

```
    LDI  N
    LD   GR1, GR0
    LDI  10
    CMP  GR0, GR1
    JC   L1
    JMP  L2
L1: 処理1
L2: 処理2
```

```
if ( n <= 10 )
```

```
    ...
    CMP  GR0, GR1
    JC   L1
    処理1
L1: 処理2
```

条件分岐のプログラム(2)

```
if ( n >= 10 ) {
    処理1
} else {
    処理2
}
処理3
```

```
    LDI  N
    LD   GR1, GR0
    LDI  10
    CMP  GR1, GR0
    JC   L1

    処理1
    JMP  L2

L1: 処理2

L2: 処理3
```

```
if ( n >= 10 ) {
    処理1
} else if ( n != 0 ) {
    処理2
} else {
    処理3
}
処理4
```

```
    LDI  N
    LD   GR1, GR0
    LDI  10
    CMP  GR1, GR0
    JC   L1

    処理1
    JMP  L3

L1: LDI  0
    CMP  GR1, GR0
    JZ   L2

    処理2
    JMP  L3

L2: 処理3

L3: 処理4
```

繰り返しのプログラム

後判定

```
do {
    処理1
} while ( n < 10 );
処理2
```

```
L1: 処理1

    LDI   N
    LD    GR1, GR0
    LDI   10
    CMP   GR1, GR0
    JC    L1

    処理2
```

前判定

```
While ( n < 10 ) {
    処理1
}
処理2
```

```
L1: LDI   N
    LD    GR1, GR0
    LDI   10
    CMP   GR1, GR0
    JC    L2
    JMP   L3

L2: 処理1
    JMP   L1

L3: 処理2
```

プログラム例(1)

1～10の総和の計算(後判定)

C言語プログラム	アセンブラプログラム	意味	動作

```
s = 0;
n = 0;
do{
  n ++;
  s += n;
} while(n != 10);
```

```
     LDI 0
     MOV GR1,GR0 ; sの初期値              GR1 = s = 0
     MOV GR2,GR0 ; nの初期値              GR2 = n = 0
     LDI 10      ;
     MOV GR3,GR0 ; nの上限値              GR3 = 10
     LDI 1
LM:  ADD GR2,GR0 ; nに1を加算             GR2 = n + 1
     ADD GR1,GR2 ; sにnを加算             GR1 = s + n
     CMP GR2,GR3 ; nを10と比較
     JZ  LE      ; 等しければ終了         n == 10
     JMP LM      ; 等しくなければ繰り返し n != 10
LE:
```

```
s = 0;
n = 0;
do{
  n ++;
  s += n;
} while(n < 10);
```

```
     ...
LM:  ADD GR2,GR0 ; nに1を加算             GR2 = n + 1
     ADD GR1,GR2 ; sにnを加算             GR1 = s + n
     CMP GR2,GR3 ; nを10と比較
     JC  LM      ; 小さければ繰り返し      n < 10
```

プログラム例(2)

1～10の総和の計算(前判定)

```
s = 0;
n = 0;
while(n != 10){
  n ++;
  s += n;
}
```

```
    LDI 0
    MOV GR1,GR0 ; sの初期値              GR1 = s = 0
    MOV GR2,GR0 ; nの初期値              GR2 = n = 0
    LDI 10      ;
    MOV GR3,GR0 ; nの上限値              GR3 = 10
    LDI 1
LC: CMP GR2,GR3 ; nを10と比較
    JZ  LE      ; 等しければ終了         n == 10
    ADD GR2,GR0 ; nに1を加算             GR2 = n + 1
    ADD GR1,GR2 ; sにnを加算             GR1 = s + n
    JMP LC      ; 繰り返し
LE:
```

```
s = 0;
n = 0;
while(n < 10){
  n ++;
  s += n;
}
```

```
    ...
LC: CMP GR2,GR3 ; nを10と比較
    JC  LM      ; 小さければ繰り返し     n < 10
    JMP LE      ; 小さくなければ終了     n >= 10
LM: ADD GR2,GR0 ; nに1を加算             GR2 = n + 1
    ADD GR1,GR2 ; sにnを加算             GR1 = s + n
    JMP LC      ; 繰り返し
LE:
```

機械語プログラム（メモリの内容）

命令メモリ（IM)

```
ADRS      INST
0000 : 0000 0000  ;       LDI  0
0001 : 0001 01 00 ;       MOV  GR1,GR0
0010 : 0001 10 00 ;       MOV  GR2,GR0
0011 : 0000 1010  ;       LDI  10
0100 : 0001 11 00 ;       MOV  GR3,GR0
0101 : 0000 0001  ;       LDI  1
0110 : 0111 10 11 ;  LC:  CMP  GR2,GR3
0111 : 1000 1011  ;       JZ   LE
1000 : 0010 10 00 ;       ADD  GR2,GR0
1001 : 0010 01 10 ;       ADD  GR1,GR2
1010 : 0110 0110  ;       JMP  LC
1011 : xxxx xxxx  ;  LE:
1100 : xxxx xxxx  ;
1101 : xxxx xxxx  ;
1110 : xxxx xxxx  ;
1111 : xxxx xxxx  ;
```

データメモリ（DM)

```
ADRS   DATA
0000 : 0000 ;
0001 : 0000 ;
0010 : 0000 ;
0011 : 0000 ;
0100 : 0000 ;
0101 : 0000 ;
0110 : 0000 ;
0111 : 0000 ;
1000 : 0000 ;
1001 : 0000 ;
1010 : 0000 ;
1011 : 0000 ;
1100 : 0000 ;
1101 : 0000 ;
1110 : 0000 ;
1111 : 0000 ;
```

命令メモリ（IM）　・・・　アドレス：4bit（0～15番地）　命令：8bit
データメモリ（DM）・・・　アドレス：4bit（0～15番地）　データ：4bit
ジャンプ先のラベル LC と LE は実際のジャンプ先番地と対応

課題

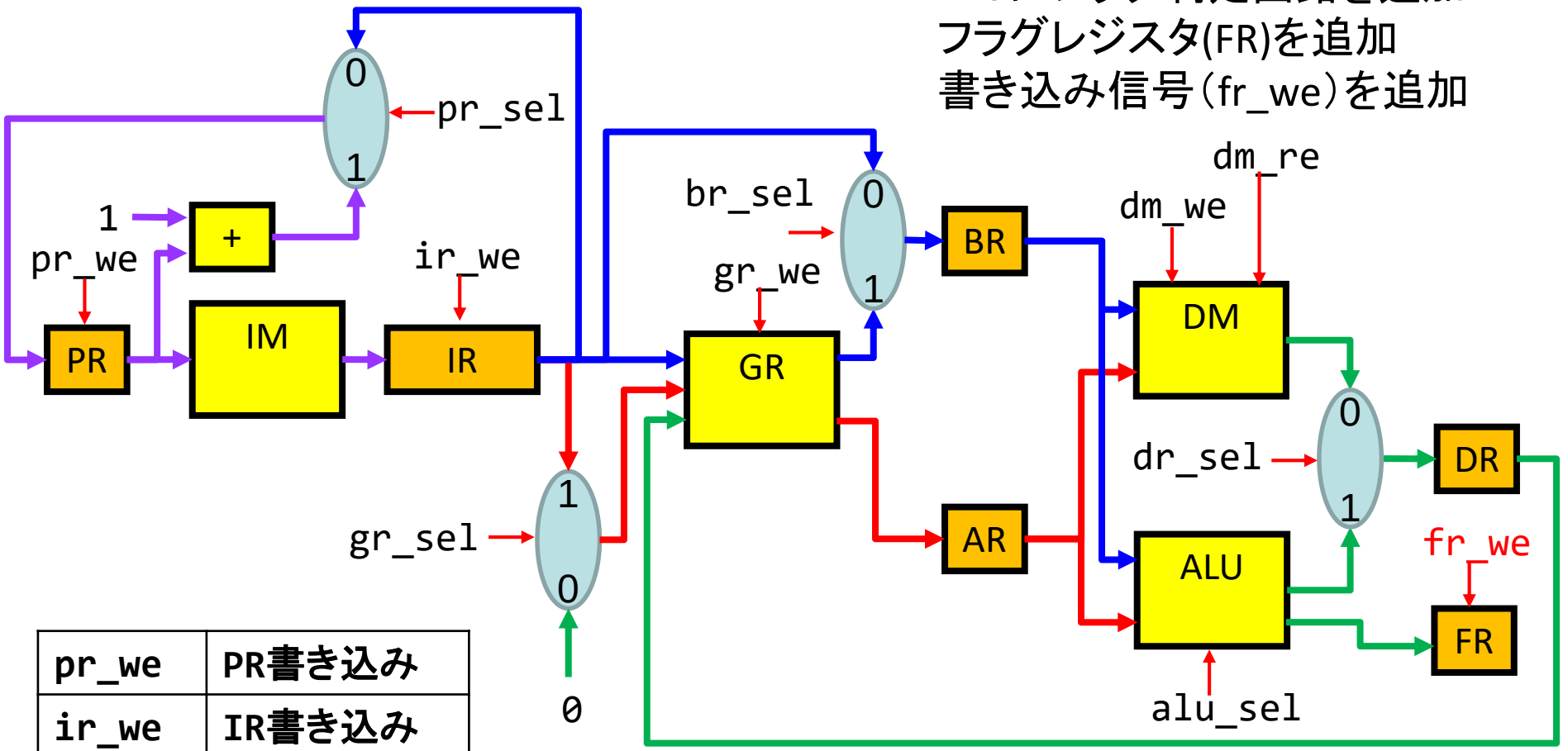
（1）以下の命令でフラグが設定されるように変更を加えよ。

ADD　加算
SUB　減算
CMP　比較

時刻	ADD (0010)	SUB (0011)	CMP (0111)
0	IR ← IM[PR] PR ← PR+1	IR ← IM[PR] PR ← PR+1	IR ← IM[PR] PR ← PR+1
1	AR ← GRa BR ← GRb	AR ← GRa BR ← GRb	AR ← GRa BR ← GRb
2	DR ← AR + BR FR ← {CF, ZF}	DR ← AR − BR FR ← {CF, ZF}	AR − BR FR ← {CF, ZF}
3	GRa ← DR	GRa ← DR	

フラグ判定回路の追加

ALUにフラグ判定回路を追加
フラグレジスタ(FR)を追加
書き込み信号(fr_we)を追加



pr_we	PR書き込み
ir_we	IR書き込み
gr_we	GR書き込み
fr_we	FR書き込み
dm_we	DM書き込み
dm_re	DM読み出し

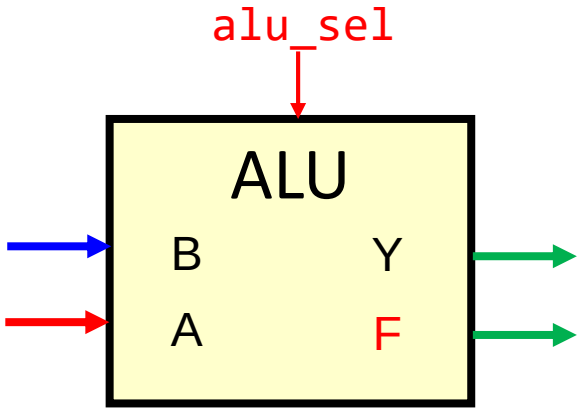
pr_sel	PR+1 または IR[3:0]
gr_sel	IR[3:2] または 0
br_sel	GRb または IR[3:0]
dr_sel	ALU または DM

alu_sel	1	B
	2	A+B
	3	A-B

ALUの変更

```
module alu(
  a, b, sel, y, f
);
  input  [3:0] a, b;
  input  [2:0] sel;
  output [3:0] y;
  output [1:0] f; /* フラグ */
  reg    [3:0] y;
  reg          c; /* キャリー */

  always@(a or b or sel) begin
    case(sel)
      ...
      ...
      ...
      ...
      default: ...;
    endcase
  end
  assign f[0] = ...; /* ZF */
  assign f[1] = ...; /* CF */
endmodule
```




```
  0111        1111
+ 0001      + 0001
------      ------
 01000       10000
 {0,0}       {1,1}

  1000        1000        0000
- 0001      - 1000      - 0001
------      ------      ------
 00111       00000       11111
 {0,0}       {0,1}       {1,0}
```

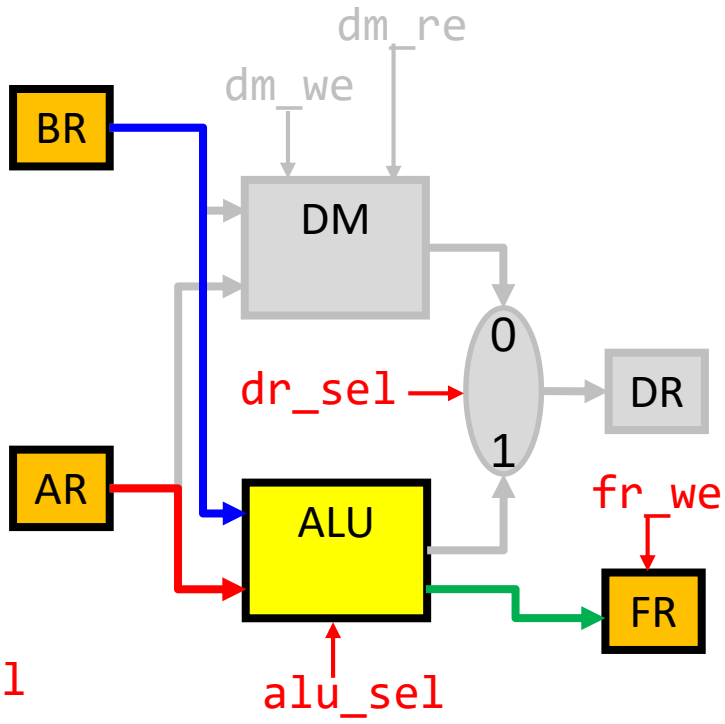
演算結果とキャリーからフラグを生成

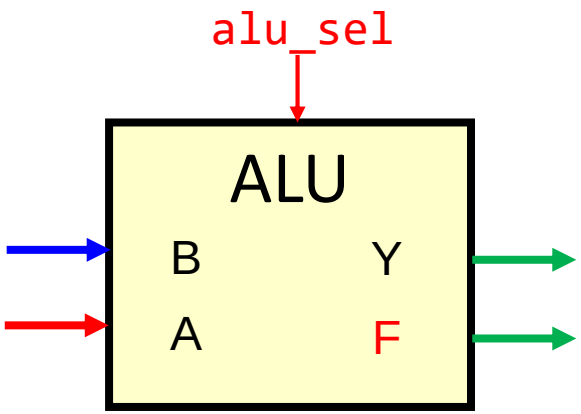
FRの追加

```
assign alu_din1 = ar_dout;
assign alu_din2 = br_dout;
alu alu(.a(alu_din1), .b(alu_din2),
        .sel(alu_sel), .y(alu_dout),
        .f(alu_fout));
```

```
assign fr_din = alu_fout;
reg2 fr(.clk(clk), .rst(1'b1), .we(fr_we),
        .din(fr_din), .dout(fr_dout));
```

ALUにフラグ判定回路を追加
フラグレジスタ(FR)を追加
書き込み信号（fr_we）を追加

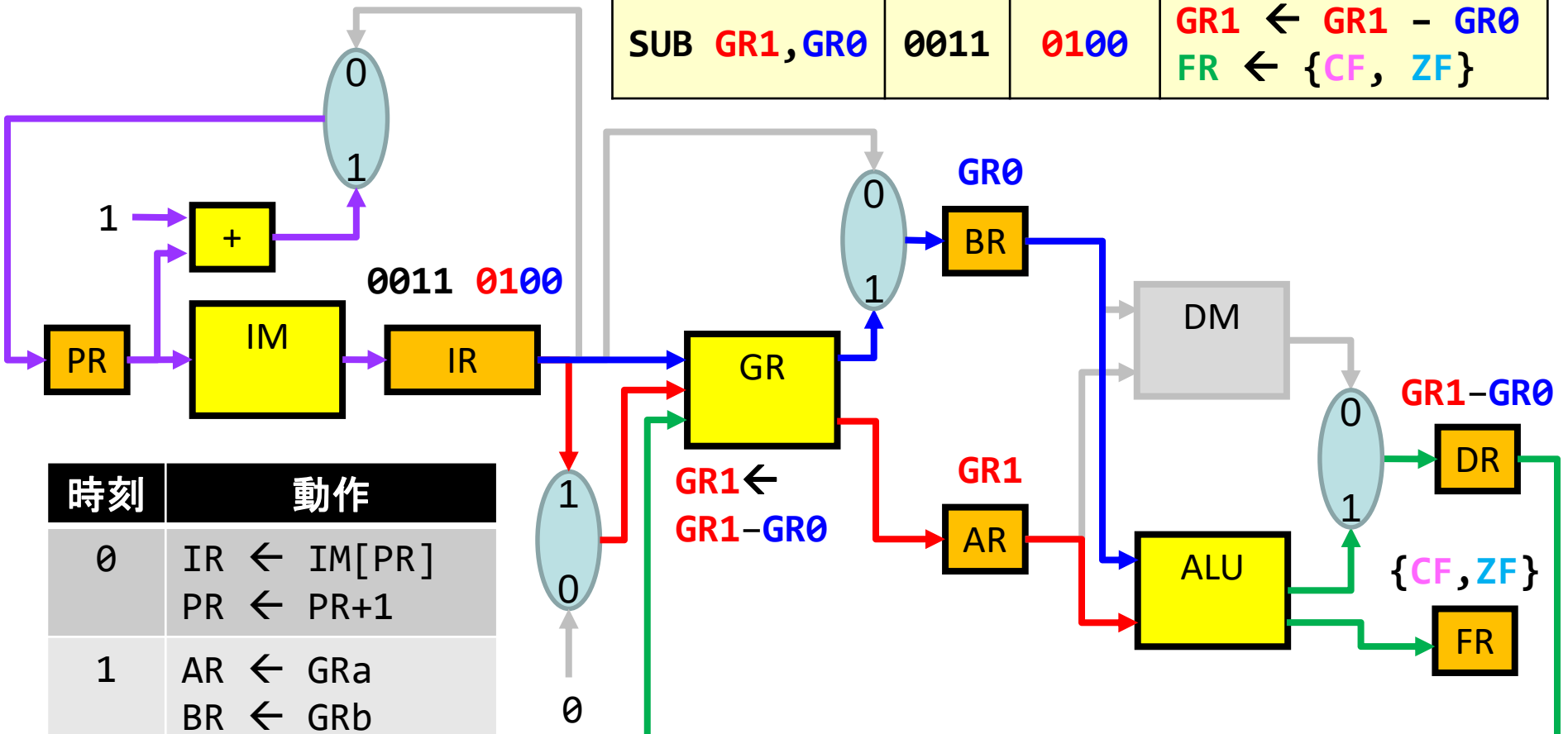




alu_sel	1	B
	2	A+B
	3	A-B

SUB命令の動作

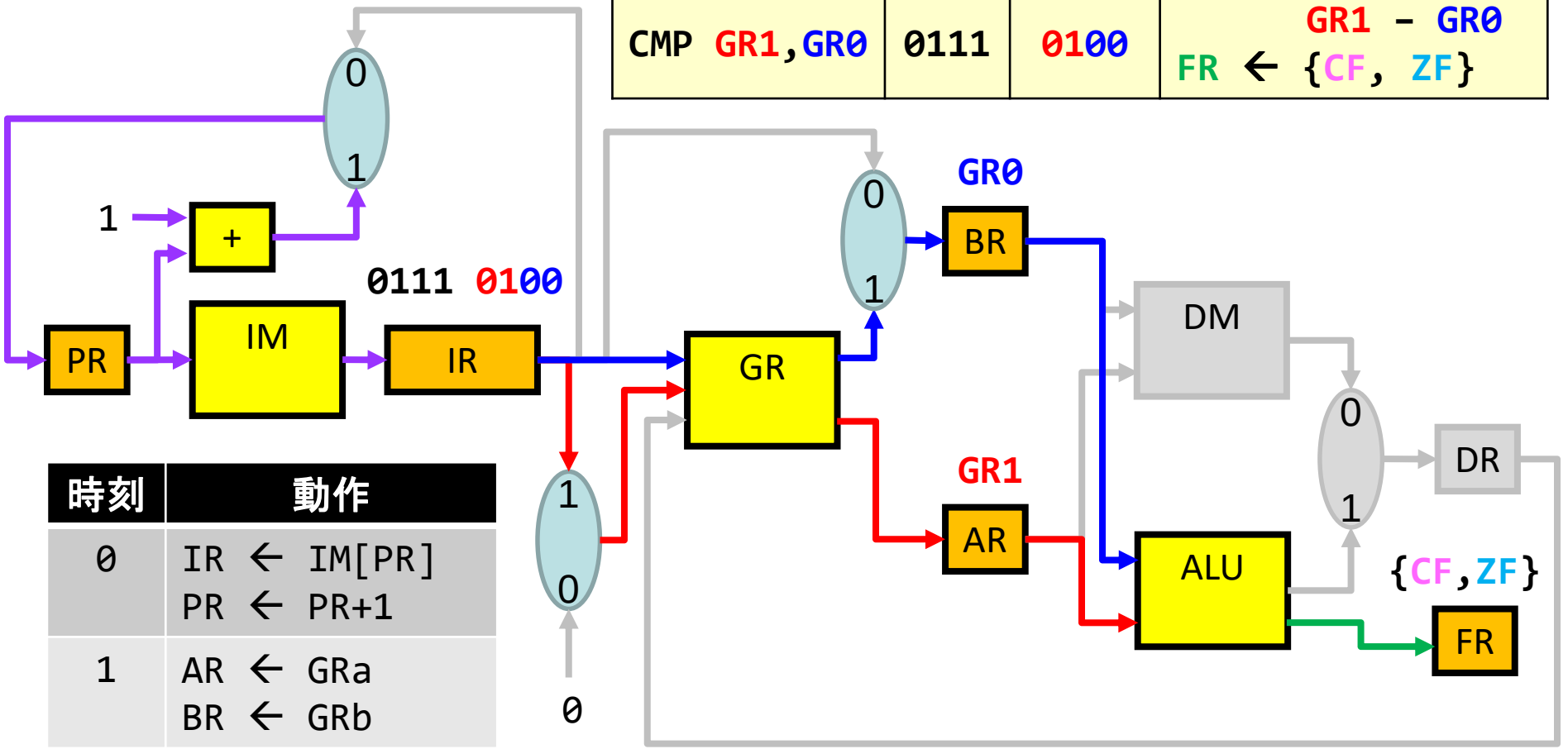
SUB GR1,GR0	0011	0100	GR1 ← GR1 - GR0 FR ← {CF, ZF}



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	AR ← GRa BR ← GRb
2	DR ← AR - BR FR ← {CF,ZF}
3	GRa ← DR

CMP命令の動作

CMP GR1,GR0	0111	0100	GR1 - GR0 FR ← {CF, ZF}



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	AR ← GRa BR ← GRb
2	AR - BR FR ← {CF,ZF}

各命令のRTL動作と制御信号

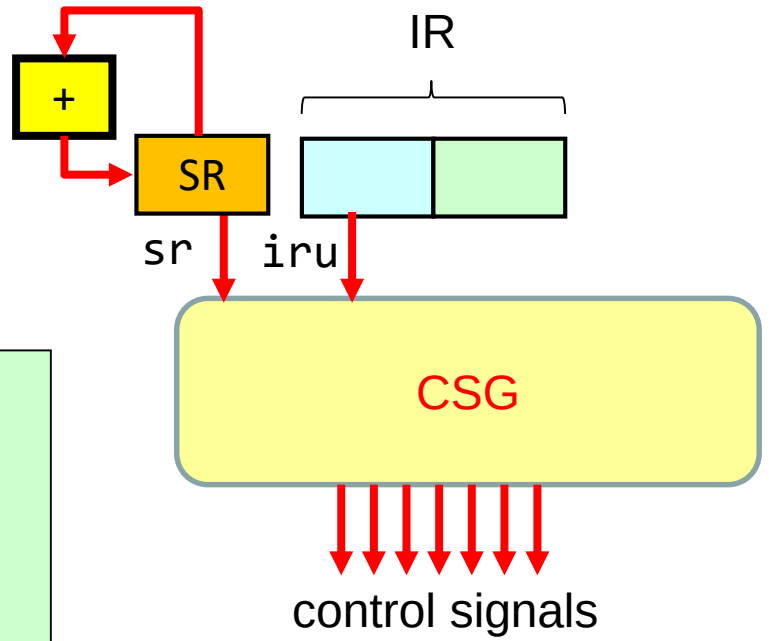
時刻	ADD (0010)	SUB (0011)	CMP (0111)
0	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1
1	AR ← GRa BR ← GRb gr_sel = 1 br_sel = 1	AR ← GRa BR ← GRb gr_sel = 1 br_sel = 1	AR ← GRa BR ← GRb gr_sel = 1 br_sel = 1
2	DR ← AR + BR FR ← {CF, ZF} alu_sel = 10 dr_sel = 1 fr_we = 1	DR ← AR – BR FR ← {CF, ZF} alu_sel = 11 dr_sel = 1 fr_we = 1	AR – BR FR ← {CF, ZF} alu_sel = 11 fr_we = 1
3	GRa ← DR gr_sel = 1 gr_we = 1	GRa ← DR gr_sel = 1 gr_we = 1	

CMPは減算して
フラグのみ格納

FR書き込み信号の追加

```
module csg(
  iru, sr,
  pr_sel, gr_sel, br_sel, dr_sel,
  alu_sel,
  ir_we, pr_we, gr_we, fr_we,
  dm_re, dm_we
);
...
```

```
always@(iru or sr) begin
  pr_sel  <= 1'bx;
  gr_sel  <= 1'bx;
  br_sel  <= 1'bx;
  dr_sel  <= 1'bx;
  alu_sel <= 2'bxx;
  ir_we   <= 1'b0;
  pr_we   <= 1'b0;
  gr_we   <= 1'b0;
  fr_we   <= 1'b0;
  dm_re   <= 1'b0;
  dm_we   <= 1'b0;
  case(sr)
    ...
  endcase
end
```



IR：命令レジスタ
SR：ステートレジスタ
CSG：制御信号生成回路

書き込み信号（fr_we）を追加
ADD, SUBの制御を修正
CMPの制御を追加

課題

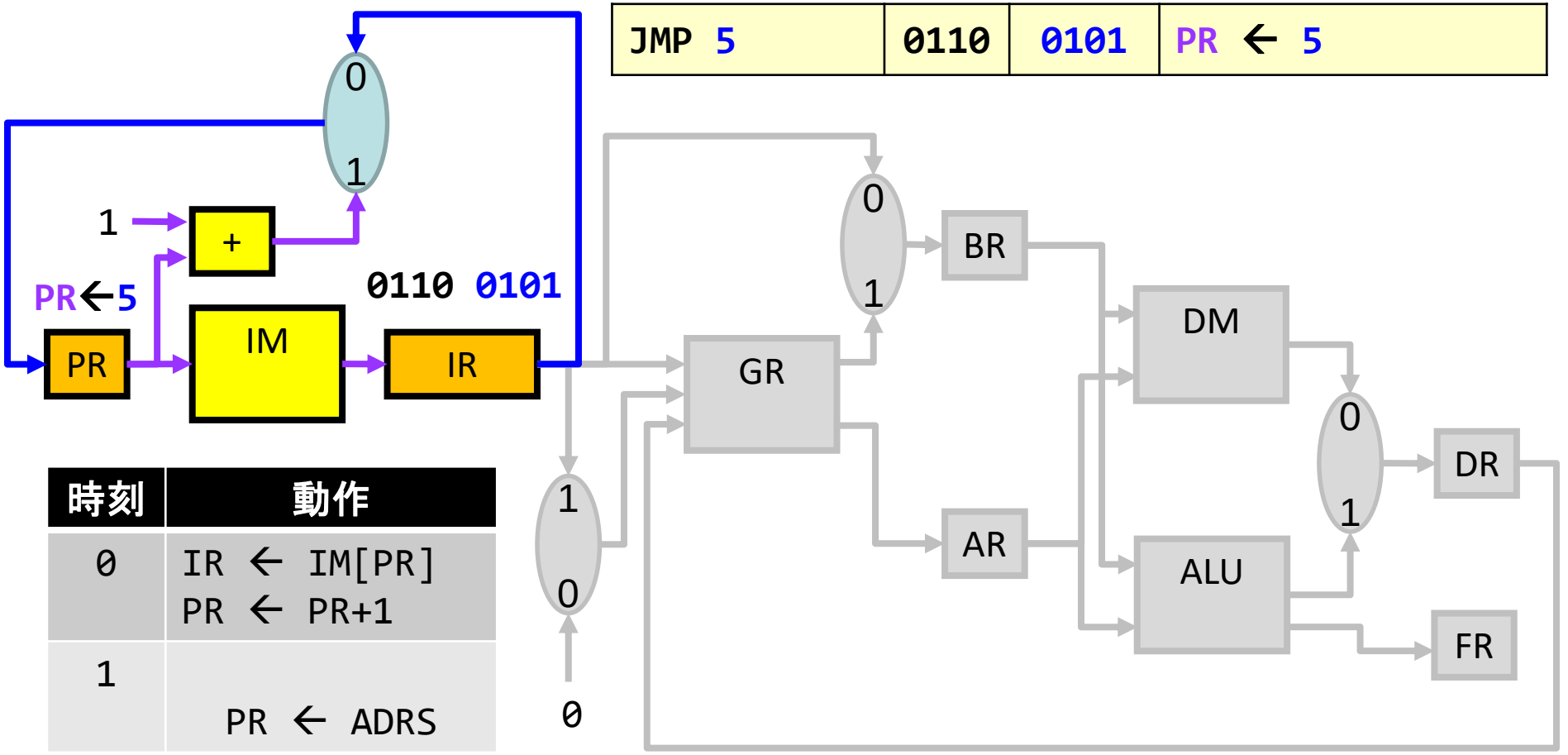
（2）以下の命令を実行できるようにCPUに変更を加えよ。

JZ　　ゼロジャンプ
JC　　キャリージャンプ

時刻	JMP (0110)	JZ (1000)	JC (1001)
0	IR ← IM[PR] PR ← PR+1	IR ← IM[PR] PR ← PR+1	IR ← IM[PR] PR ← PR+1
1	PR ← ADRS	if(FR[0]==1) PR ← ADRS	if(FR[1]==1) PR ← ADRS

JMP命令の動作

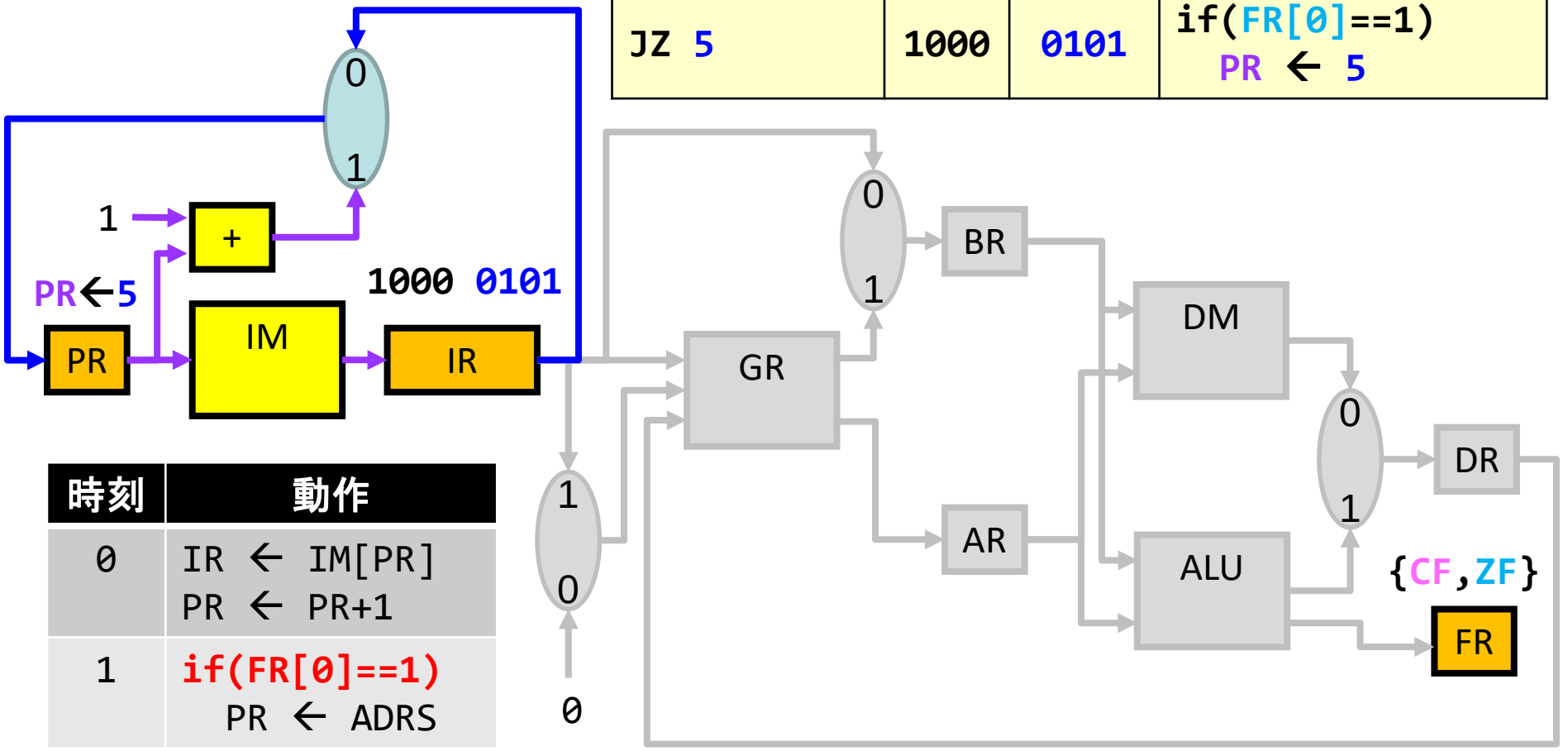
JMP 5	0110	0101	PR ← 5



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	PR ← ADRS

JZ命令の動作

JZ 5	1000	0101	if(FR[0]==1) PR ← 5



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	if(FR[0]==1) PR ← ADRS

各命令のRTL動作と制御信号

時刻	JMP (1010)	JZ (1000)	JC (1001)
0	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1
1	PR ← ADRS pr_sel = 0 pr_we = 1	if(FR[0]==1) PR ← ADRS pr_sel = 0 pr_we = FR[0]	if(FR[1]==1) PR ← ADRS pr_sel = 0 pr_we = FR[1]

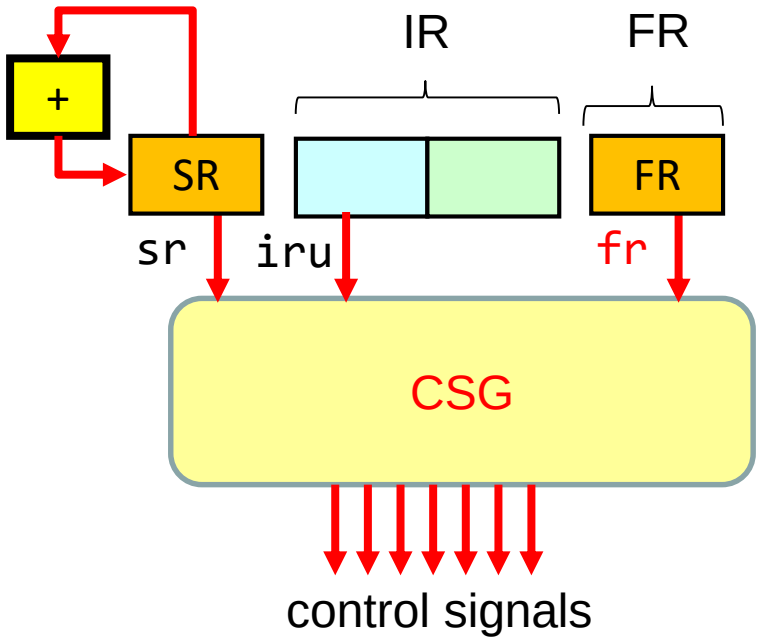
条件ジャンプは時刻1で
フラグの条件に応じてPRを変更

FRの値をPR書き込み信号にそのまま利用可

FRの入力を追加

```
module csg(
  iru, sr, fr,
  pr_sel, gr_sel, br_sel, dr_sel,
  alu_sel,
  ir_we, pr_we, gr_we, fr_we,
  dm_re, dm_we
);
...
```

```
always@(iru or sr or fr) begin
  pr_sel  <= 1'bx;
  gr_sel  <= 1'bx;
  br_sel  <= 1'bx;
  dr_sel  <= 1'bx;
  alu_sel <= 2'bxx;
  ir_we   <= 1'b0;
  pr_we   <= 1'b0;
  gr_we   <= 1'b0;
  fr_we   <= 1'b0;
  dm_re   <= 1'b0;
  dm_we   <= 1'b0;
  case(sr)
    ...
  endcase
end
```



IR：命令レジスタ
SR：ステートレジスタ
CSG：制御信号生成回路

フラグレジスタ（FR）をCSGに入力
JZ, JCの制御を追加

即値演算命令

命令	ニモニック	操作コード	動作	例
即値ロード	LDI ADRS	0 (0000)	GR0 ← IMM	*p = 6; y = 6
加算	ADD GRa, GRb	2 (0010)	GRa ← GRa + GRb FR ← {CF, ZF}	y += x;
比較	CMP GRa, GRb	7 (0111)	GRa - GRb FR ← {CF, ZF}	y - x
即値加算	ADDI IMM	10 (1010)	GR0 ← GR0 + IMM FR ← {CF, ZF}	y += 6;
即値比較	CMPI IMM	11 (1011)	GR0 - IMM FR ← {CF, ZF}	y - x

※ 即値ロード、即値加算、即値比較の演算対象はGR0のみ

※ 即値減算は負の値の即値加算で代用可能（キャリーは異なる）

```
  1000      1000          0000      0000
- 0001    + 1111        - 0001    + 1111
------    ------        ------    ------
 00111     10111         11111     01111
```

プログラム例(3)

1～10の総和の計算(前判定)

```
s = 0;
n = 0;
while(n != 10){
  n ++;
  s += n;
}
```

```
    LDI 0
    MOV GR1,GR0 ; sの初期値          GR1 = s = 0
    MOV GR2,GR0 ; nの初期値          GR2 = n = 0
    LDI 10      ;
    MOV GR3,GR0 ; nの上限値          GR3 = 10
    LDI 1
LC: CMP GR2,GR3 ; nを10と比較
    JZ  LE      ; 等しければ終了     n == 10
    ADD GR2,GR0 ; nに1を加算         GR2 = n + 1
    ADD GR1,GR2 ; sにnを加算         GR1 = s + n
    JMP LC      ; 繰り返し
LE:
```

nの加算値1
nの上限値10
➔格納の必要なし

レジスタ数削減
命令数削減

```
    LDI  0       ; nの初期値          GR0 = n = 0
    MOV  GR1,GR0 ; sの初期値          GR1 = s = 0
LC: CMPI 10      ; nを10と比較
    JZ   LE      ; 等しければ終了     n == 10
    ADDI 1       ; nに1を加算         GR0 = n + 1
    ADD  GR1,GR0 ; sにnを加算         GR1 = s + n
    JMP  LC      ; 繰り返し
LE:
```

プログラム例(4)

4番地～7番地に格納された値の総和の計算(後判定)

```
s = 0;
a = 4;
do{
  s += *a;
  a ++;
} while( a < 8 );
```

```
    LDI  0        ;
    MOV  GR1,GR0  ; sの初期値          GR1 = s = 0
    LDI  4        ; aの初期値          GR0 = a = 4
LM: LD   GR2,GR0  ; a番地の内容をロード
    ADD  GR1,GR2  ; ロードした値を加算   GR1 = s += *a
    ADDI 1        ; aに1を加算         GR0 = a ++
    CMPI 8        ; aを8と比較
    JC   LM       ; 小さければ繰り返し   a < 8
LE:
```

データメモリ(DM)

```
ADRS    DATA
0000 : 0111 ; 7
0001 : 0011 ; 3
0010 : 0100 ; 4
0011 : 0001 ; 1
0100 : 0110 ; 6
0101 : 0011 ; 3
0110 : 0010 ; 2
0111 : 0101 ; 5
1000 : 0100 ; 4
```

実行例

```
 GR0   GR1   GR2   命令
 4     0     x     LD   GR2,GR0
 4     0     6     ADD  GR1,GR2
 4     6     6     ADDI 1
 5     6     6     CMPI 8
...
 5     6     6     LD   GR2,GR0
 5     6     3     ADD  GR1,GR2
 5     9     3     ADDI 1
 6     9     3     CMPI 8
...
```

課題

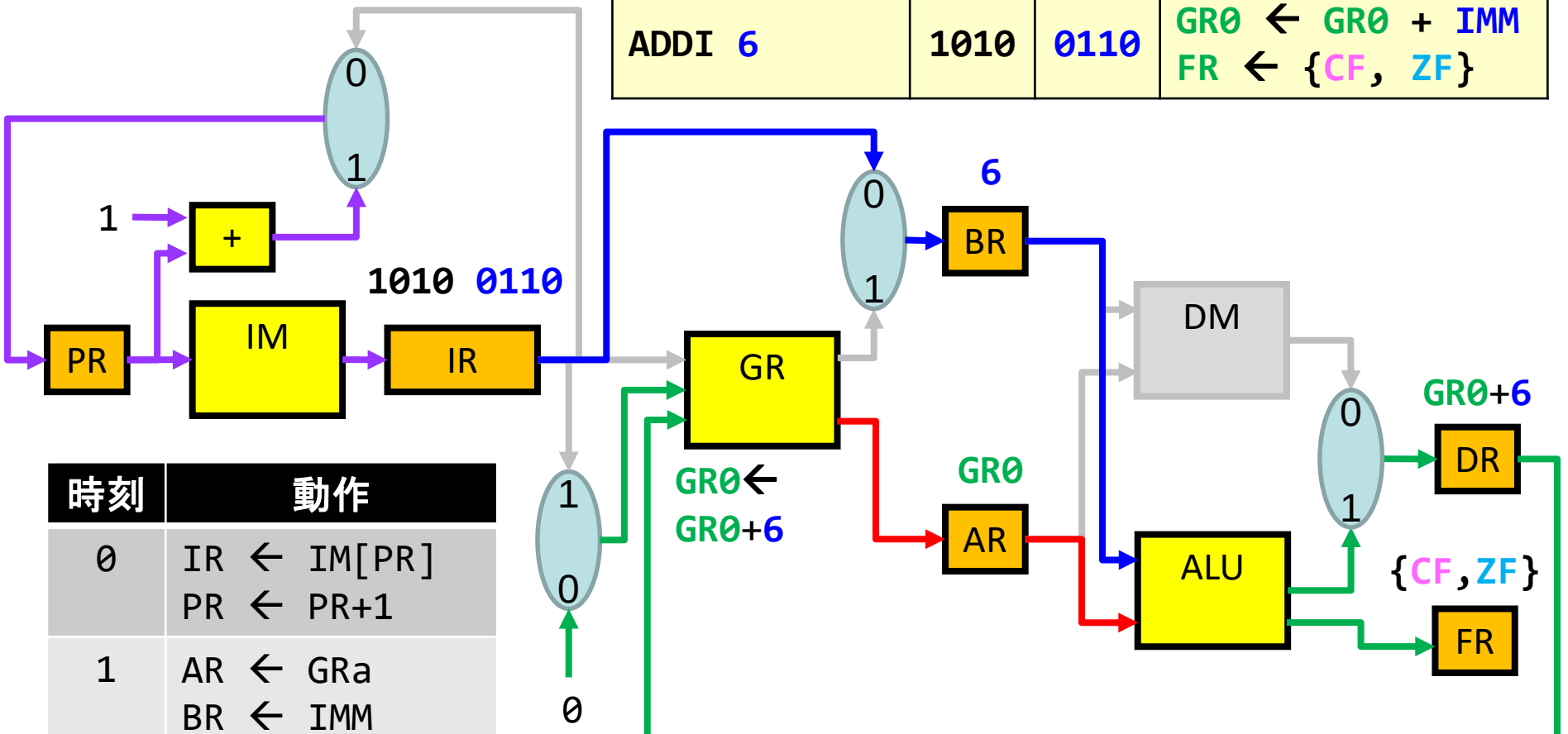
（3）以下の命令を実行できるようにCPUに変更を加えよ。

ADDI　即値加算
CMPI　即値比較

時刻	ADD (0010)	CMP (0111)	ADDI (1010)	CMPI (1011)
0	IR ← IM[PR] PR ← PR+1	IR ← IM[PR] PR ← PR+1	IR ← IM[PR] PR ← PR+1	IR ← IM[PR] PR ← PR+1
1	AR ← GRa BR ← GRb	AR ← GRa BR ← GRb	AR ← GR0 BR ← IMM	AR ← GR0 BR ← IMM
2	DR ← AR + BR FR ← {CF, ZF}	AR − BR FR ← {CF, ZF}	DR ← AR + BR FR ← {CF, ZF}	AR − BR FR ← {CF, ZF}
3	GRa ← DR		GR0 ← DR	

ADDI命令の動作

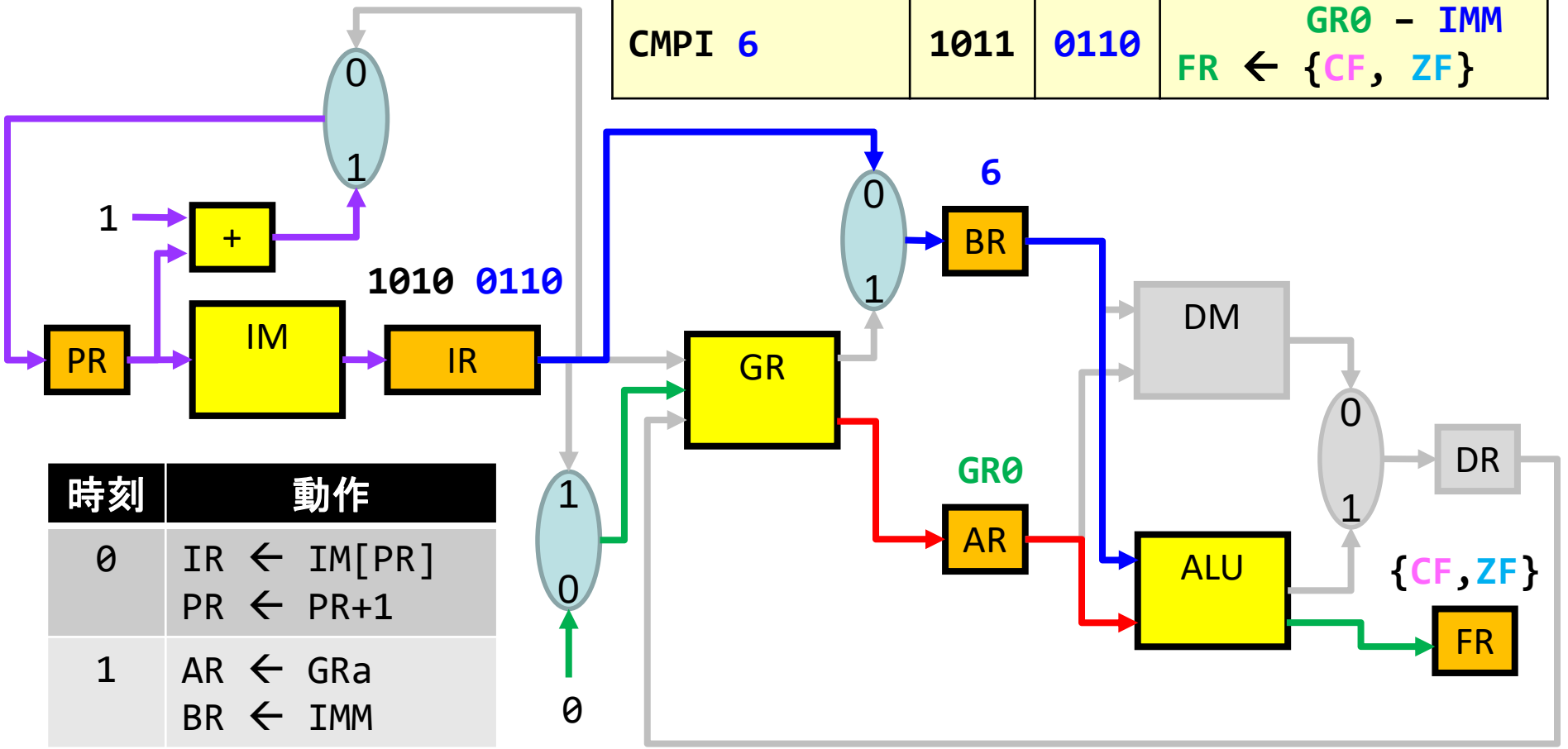
ADDI 6	1010	0110	GR0 ← GR0 + IMM FR ← {CF, ZF}



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	AR ← GRa BR ← IMM
2	DR ← AR + BR FR ← {CF,ZF}
3	GRa ← DR

CMPI命令の動作

CMPI 6	1011	0110	GR0 - IMM FR ← {CF, ZF}



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	AR ← GRa BR ← IMM
2	AR - BR FR ← {CF,ZF}

各命令のRTL動作と制御信号

時刻	ADD (0010)	CMP (0111)	ADDI (1010)	CMPI (1011)
0	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1
1	AR ← GRa BR ← GRb gr_sel = 1 br_sel = 1	AR ← GRa BR ← GRb gr_sel = 1 br_sel = 1	AR ← GR0 BR ← IMM gr_sel = 0 br_sel = 0	AR ← GR0 BR ← IMM gr_sel = 0 br_sel = 0
2	DR ← AR + BR FR ← {CF, ZF} alu_sel = 10 dr_sel = 1 fr_we = 1	AR – BR FR ← {CF, ZF} alu_sel = 11 fr_we = 1	DR ← AR + BR FR ← {CF, ZF} alu_sel = 10 dr_sel = 1 fr_we = 1	AR – BR FR ← {CF, ZF} alu_sel = 11 fr_we = 1
3	GRa ← DR gr_sel = 1 gr_we = 1		GR0 ← DR gr_sel = 0 gr_we = 1	

課題

（1）以下の命令でフラグが設定されるように変更を加えよ。

ADD	加算
SUB	減算
CMP	比較

（2）以下の命令を実行できるようにCPUに変更を加えよ。

JZ	ゼロジャンプ
JC	キャリージャンプ

（3）以下の命令を実行できるようにCPUに変更を加えよ。

ADDI	即値加算
CMPI	即値比較

テストプログラム例

例(1)

```
LDI  1
MOV  GR1, GR0
SUB  GR1, GR0
SUB  GR1, GR0
ADD  GR1, GR0
ADD  GR1, GR0
LDI  2
CMP  GR0, GR0
CMP  GR0, GR1
CMP  GR1, GR0
```

例(2)

```
     LDI  1
     MOV  GR1, GR0
     LDI  2
     MOV  GR2, GR0
     LDI  0
L1:  ADD  GR0, GR1
     CMP  GR0, GR2
     JC   L1
     JZ   L1
```

例(3)

```
     LDI  0
L1:  ADDI 1
     CMPI 2
     JC   L1
     JZ   L1
```

他のプログラムでテストしてもよい。

変更のポイント

- 課題1：フラグレジスタFRを追加
  - ALUにフラグ判定回路を追加
- 課題2：FRをCSGに入力
  - PR書き込み信号（pr_we）のために使用
- 共通：制御信号の動作をCSGに記述
- 共通：命令メモリにプログラムを記述

レポート提出

ソースファイル：Webから提出
（次回授業日の当日9時締切）

レポート：3号館1階知識工学部事務室へ提出
（次回授業日の当日9時締切）

レポートの内容：
iverilog の実行結果
gtkwave の波形（前回の課題と同様にペンなどで追記）
結果に対する説明や考察などを記述

レポートの表紙：
第6回ハードウェア記述言語レポート
学籍番号、氏名