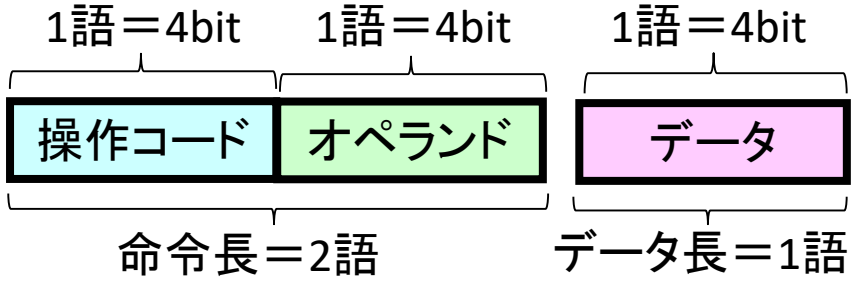
第５回 ハードウェア記述言語
～CPUのデータパス、制御信号生成回路～

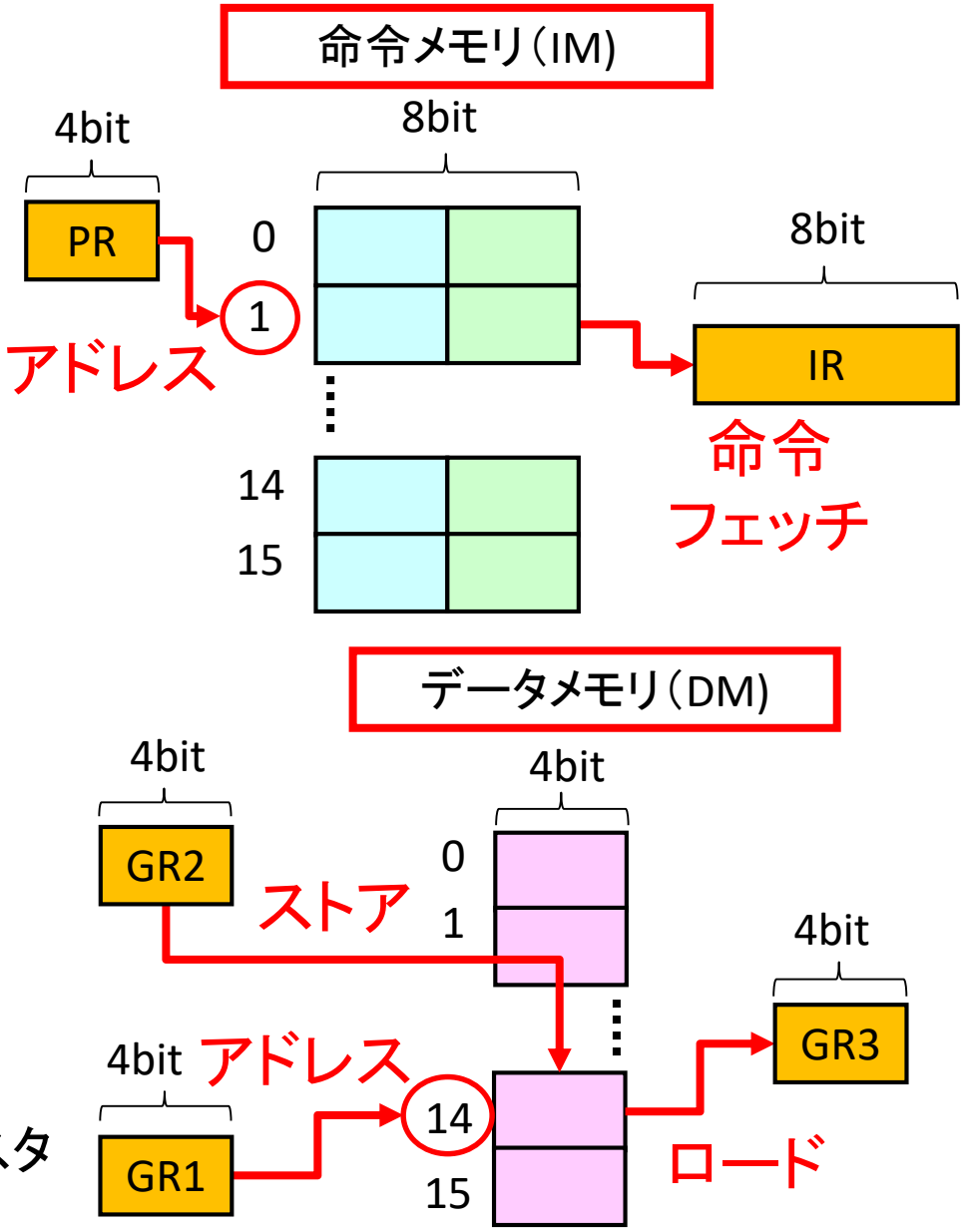
中野 秀洋

CPUの主な仕様

1語長	4bit
命令メモリ （IM） 16Byte	アドレス：4bit 0～15番地 命令長：2語＝8bit 命令：8bit ×16
データメモリ （DM) 8Byte	アドレス：4bit 0～15番地 データ長：1語=4bit データ：4bit ×16
汎用レジスタ （GR)	データ：4bit × 4 GR0～GR3



PR：プログラムレジスタ　GR：汎用レジスタ
IR：命令レジスタ



CPUの命令セット

命令	ニモニック	操作コード	動作	例
即値ロード	LDI IMM	0 (0000)	GR0 ← IMM	p = 6; y = 6
レジスタ間転送	MOV GRa, GRb	1 (0001)	GRa ← GRb	y = x;
加算	ADD GRa, GRb	2 (0010)	GRa ← GRa + GRb	y += x;
減算	SUB GRa, GRb	3 (0011)	GRa ← GRa - GRb	y -= x;
ロード	LD GRa, GRb	4 (0100)	GRa ← DM[GRb]	y = *p;
ストア	ST GRa, GRb	5 (0101)	DM[GRb] ← GRa	*p = y;
無条件ジャンプ	JMP ADRS	6 (0110)	PR ← ADRS	while(1);

※ 汎用レジスタは GR0～GR3 の4つを指定可能

※ 即値ロードの書き込み先はGR0のみ

命令と機械語の例

ニモニック	機械語 第1ワード	機械語 第2ワード	動作
LDI 6	0000	0110	GR0 ← 6
MOV GR1, GR0	0001	01 00	GR1 ← GR0
ADD GR2, GR1	0010	10 01	GR2 ← GR2 + GR1
SUB GR2, GR1	0011	10 01	GR2 ← GR2 - GR1
LD GR3, GR2	0100	11 10	GR3 ← DM[GR2]
ST GR3, GR2	0101	11 10	DM[GR2] ← GR3
JMP 5	0110	0101	PR ← 5

※ 汎用レジスタは GR0～GR3 の4つを指定可能

※ 即値ロードの操作対象は**GR0**のみ

プログラム例

C言語プログラム

```
x = 6;
do{
    y = x - 2;
    x = x + y;
while(1);
```

アセンブラプログラム

			意味	動作
	LDI	X	; Xのアドレスをロード	GR0 = &x
	MOV	GR2,GR0	; Xのアドレスを転送	GR2 = &x
	LDI	Y	; Yのアドレスをロード	GR0 = &y
	MOV	GR3,GR0	; Yのアドレスを転送	GR3 = &y
	LDI	6	; 6を即値ロード	GR0 = 6
LOOP:	ST	GR0,GR2	; Xにストア	x = 6 ; x = x + y
	MOV	GR1,GR0	; Xの値を転送	GR1 = x
	LDI	2	; 2を即値ロード	GR0 = 2
	SUB	GR1,GR0	; Xから2を減算	GR1 = x - 2
	ST	GR1,GR3	; Yにストア	y = x - 2
	LD	GR0,GR2	; Xをロード	GR0 = x
	ADD	GR0,GR1	; XにYを加算	GR0 = x + y
	JMP	LOOP	; LOOPに戻る	

機械語プログラム（メモリの内容）

命令メモリ（IM）

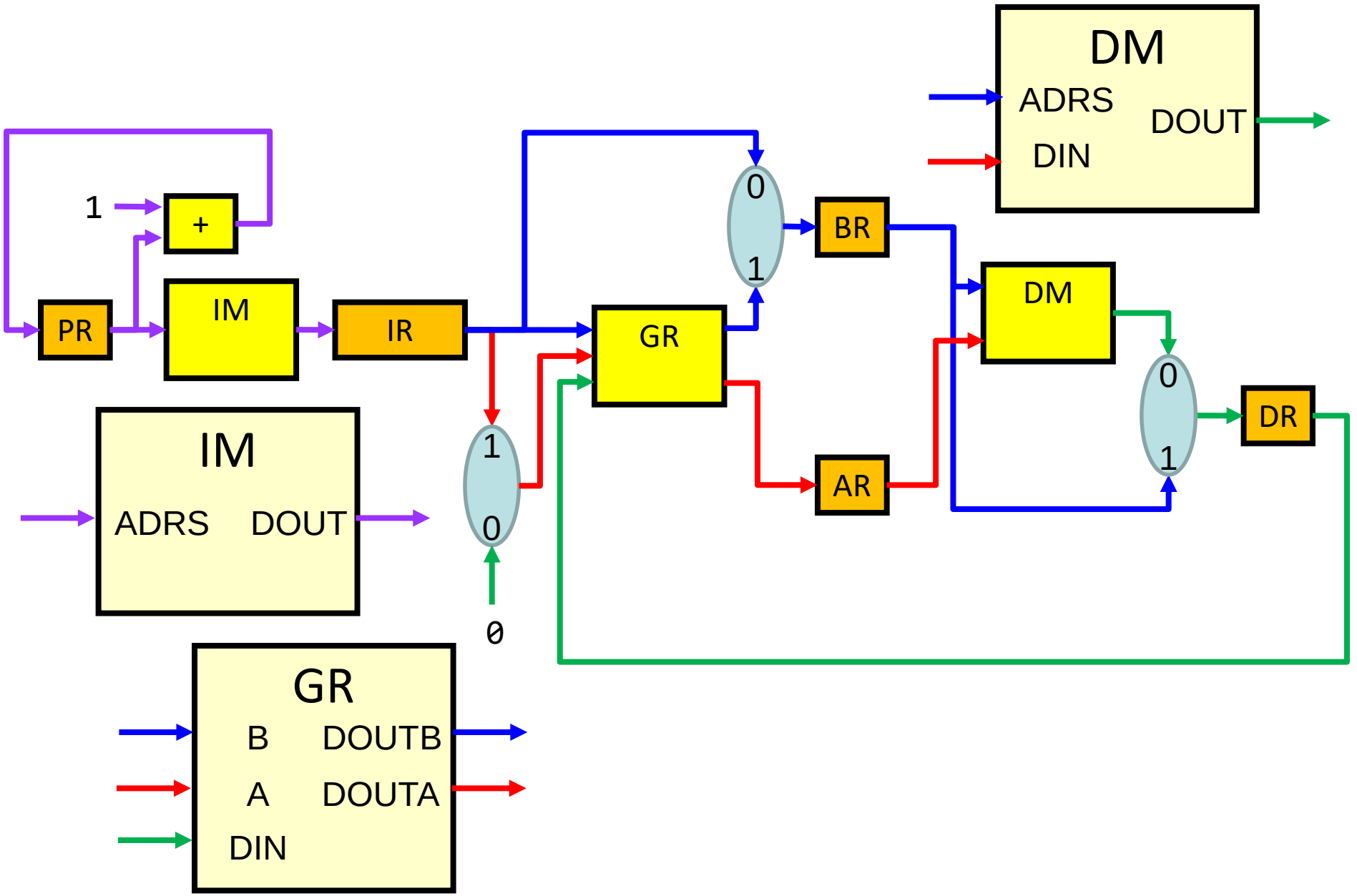
```
ADRS      INST
0000 : 0000 0000  ;        LDI   X
0001 : 0001 10 00 ;        MOV   GR2,GR0
0010 : 0000 0001  ;        LDI   Y
0011 : 0001 11 00 ;        MOV   GR3,GR0
0100 : 0000 0110  ;        LDI   6
0101 : 0101 00 10 ; LOOP:  ST    GR0,GR2
0110 : 0001 01 00 ;        MOV   GR1,GR0
0111 : 0000 0010  ;        LDI   2
1000 : 0011 01 00 ;        SUB   GR1,GR0
1001 : 0101 01 11 ;        ST    GR1,GR3
1010 : 0100 00 10 ;        LD    GR0,GR2
1011 : 0010 00 01 ;        ADD   GR0,GR1
1100 : 0110 0101  ;        JMP   LOOP
1101 : xxxx xxxx  ;
1110 : xxxx xxxx  ;
1111 : xxxx xxxx  ;
```

データメモリ（DM）

```
ADRS   DATA
0000 : 0000 ; X
0001 : 0000 ; Y
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 ;
```

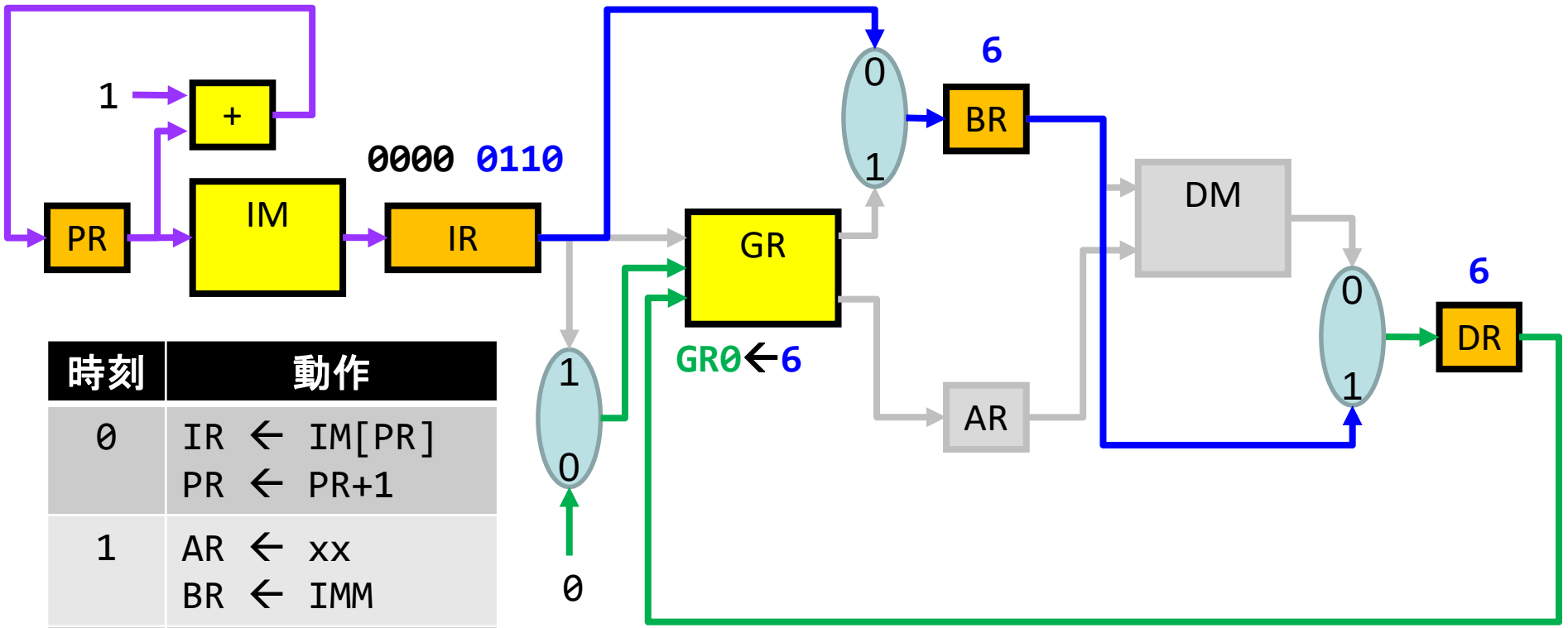
命令メモリ（IM）　・・・　アドレス：4bit（0～15番地）　命令：8bit
データメモリ（DM）・・・　アドレス：4bit（0～15番地）　データ：4bit
変数のラベル X と Y は実際の格納先番地と対応
ジャンプ先のラベル LOOP は実際のジャンプ先と対応

CPUのブロック図（初期バージョン）



LDI命令の動作

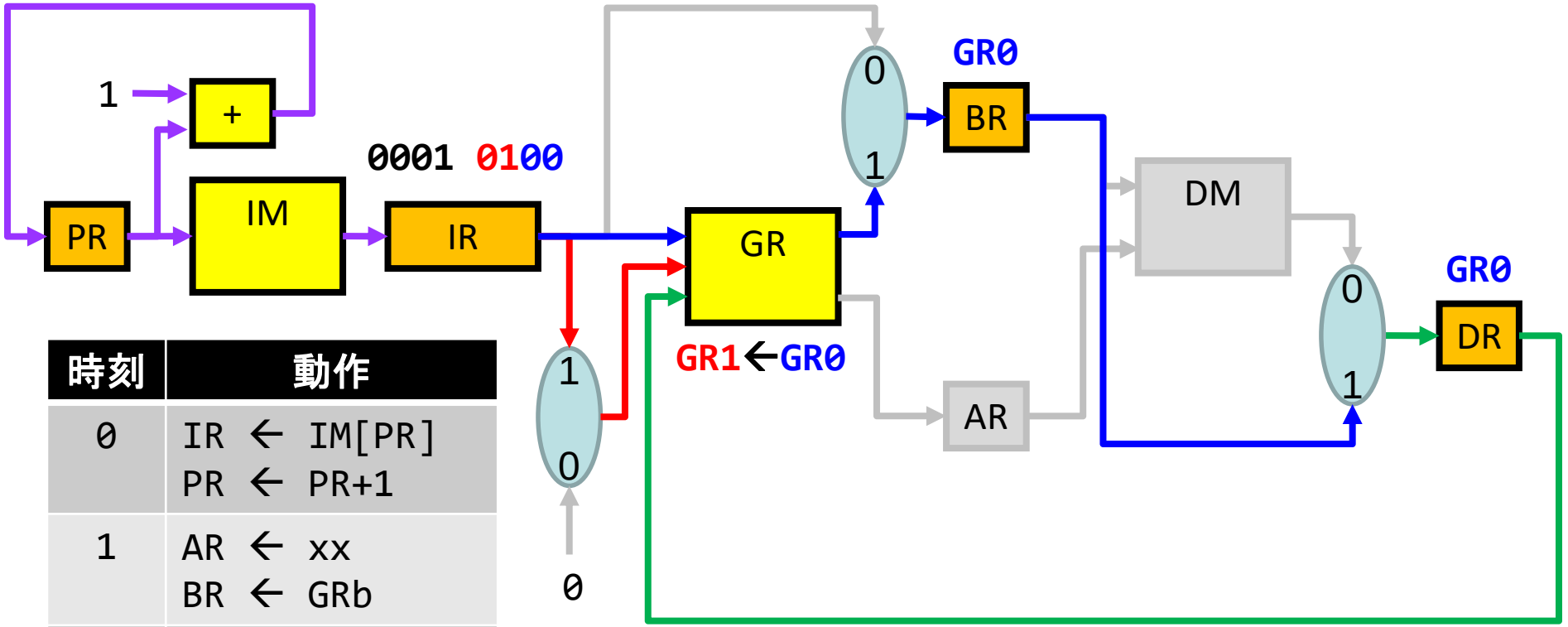
LDI 6	0000	0110	GR0 ← 6



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	AR ← xx BR ← IMM
2	DR ← BR
3	GR0 ← DR

MOV命令の動作

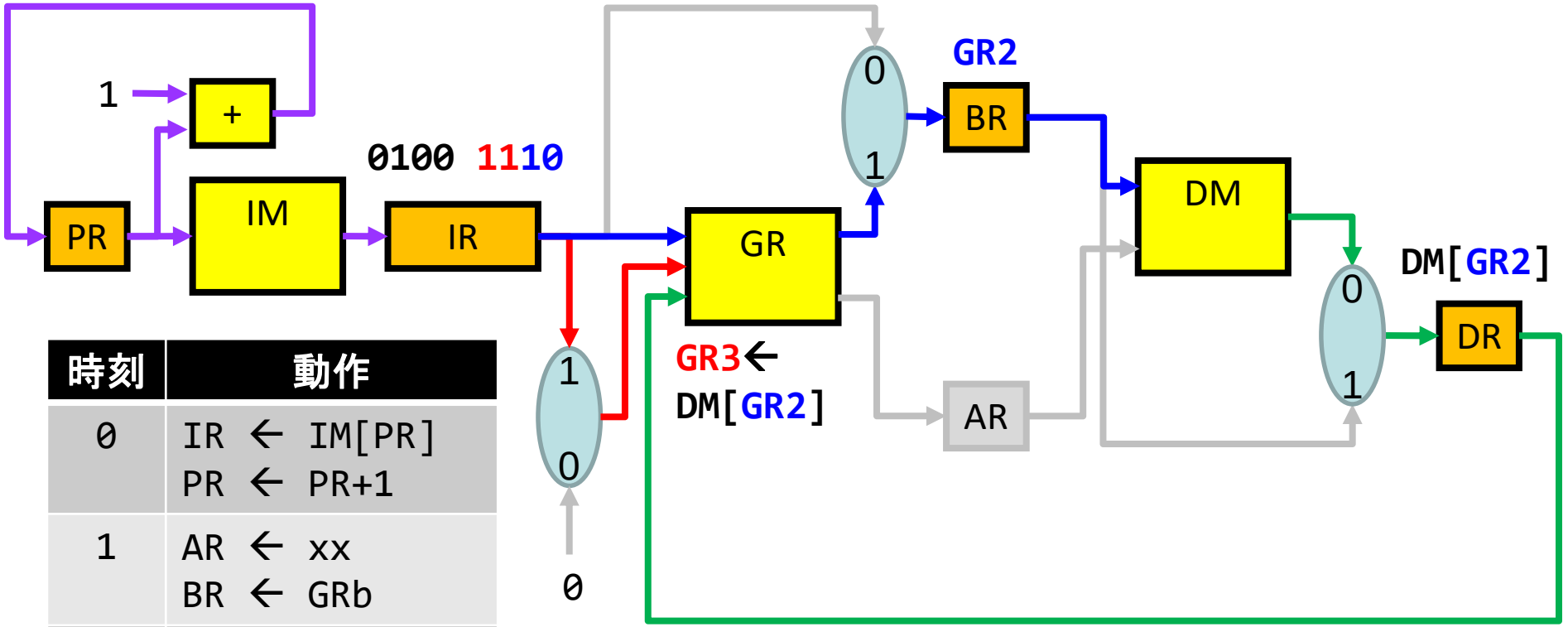
MOV GR1,GR0	0001	0100	GR1 ← GR0



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	AR ← xx BR ← GRb
2	DR ← BR
3	GRa ← DR

LD命令の動作

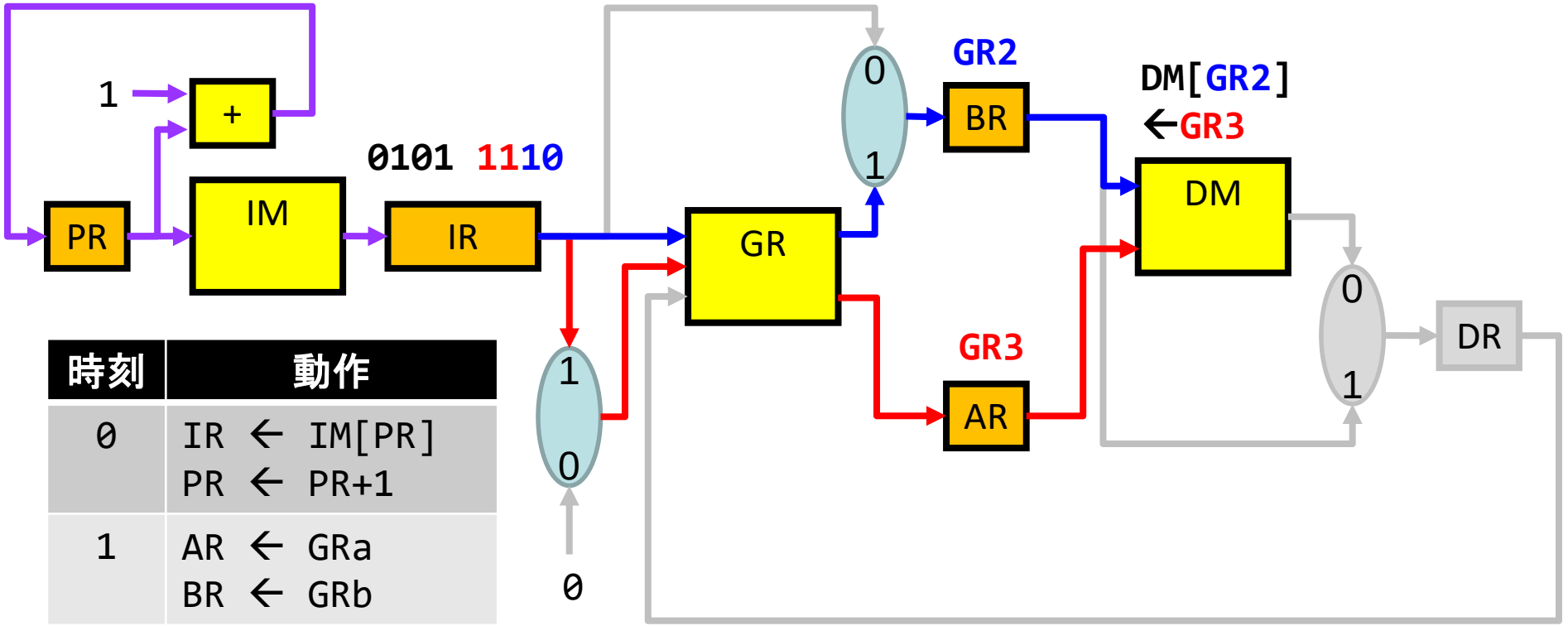
LD GR3,GR2	0100	1110	GR3 ← DM[GR2]



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	AR ← xx BR ← GRb
2	DR ← DM[BR]
3	GRa ← DR

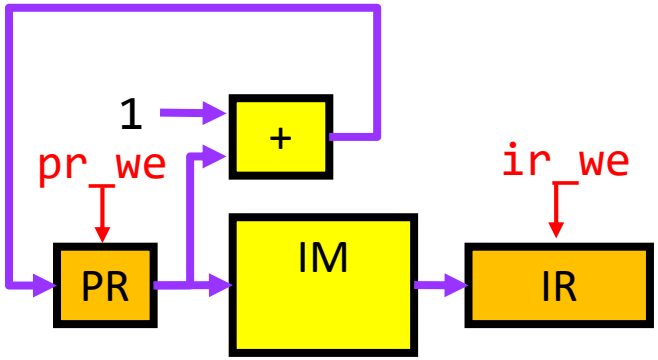
ST命令の動作

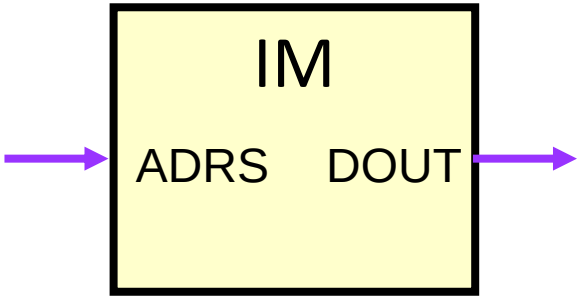
ST GR3,GR2	0101	1110	DM[GR2] ← GR3



時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	AR ← GRa BR ← GRb
2	DM[BR] ← AR

命令フェッチ：Instruction Fetch (IF)






```
assign pr_din = pr_dout + 4'b0001;
reg4 pr(.clk(clk), .rst(rst), .we(pr_we),
        .din(pr_din), .dout(pr_dout));
```

書き込み信号（pr_we）が1なら
PRを1繰り上げる

```
assign im_adrs = pr_dout;
rom im(.adrs(im_adrs), .dout(im_dout));
```

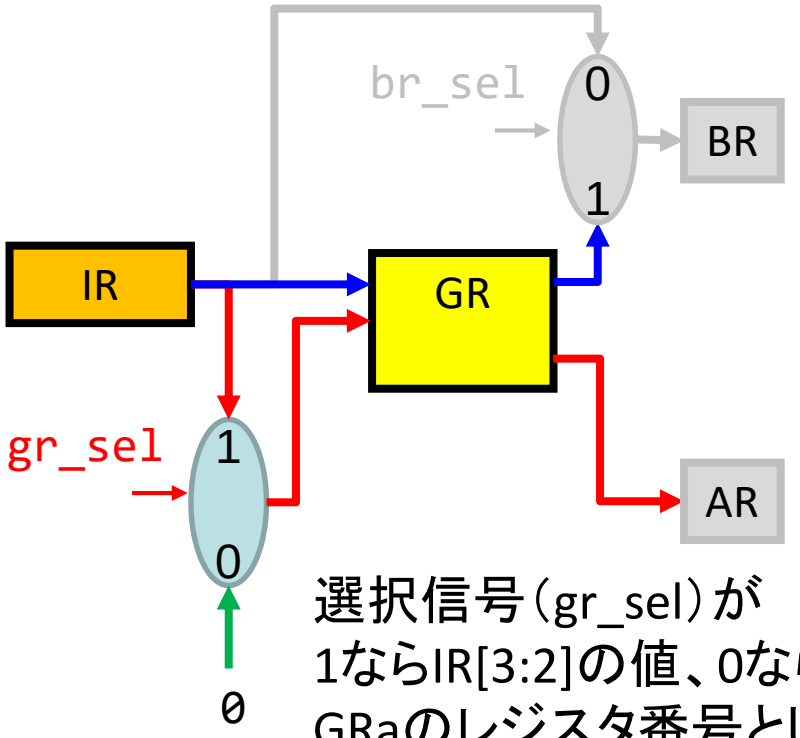
PRで指定した番地の内容（命令）を
IMから読み出す

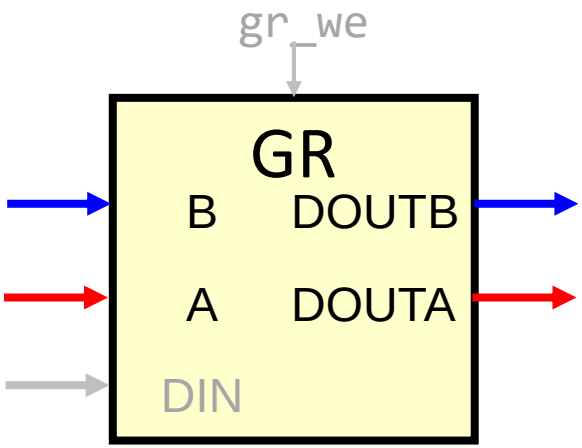
```
assign ir_din = im_dout;
reg8 ir(.clk(clk), .rst(rst), .we(ir_we),
        .din(ir_din), .dout(ir_dout));
```

書き込み信号（ir_we）が1なら
IMの内容（命令）をIRに格納する

レジスタ・フェッチ： Register Fetch (RF)

IR[1:0]の値を
GRbのレジスタ番号として指定

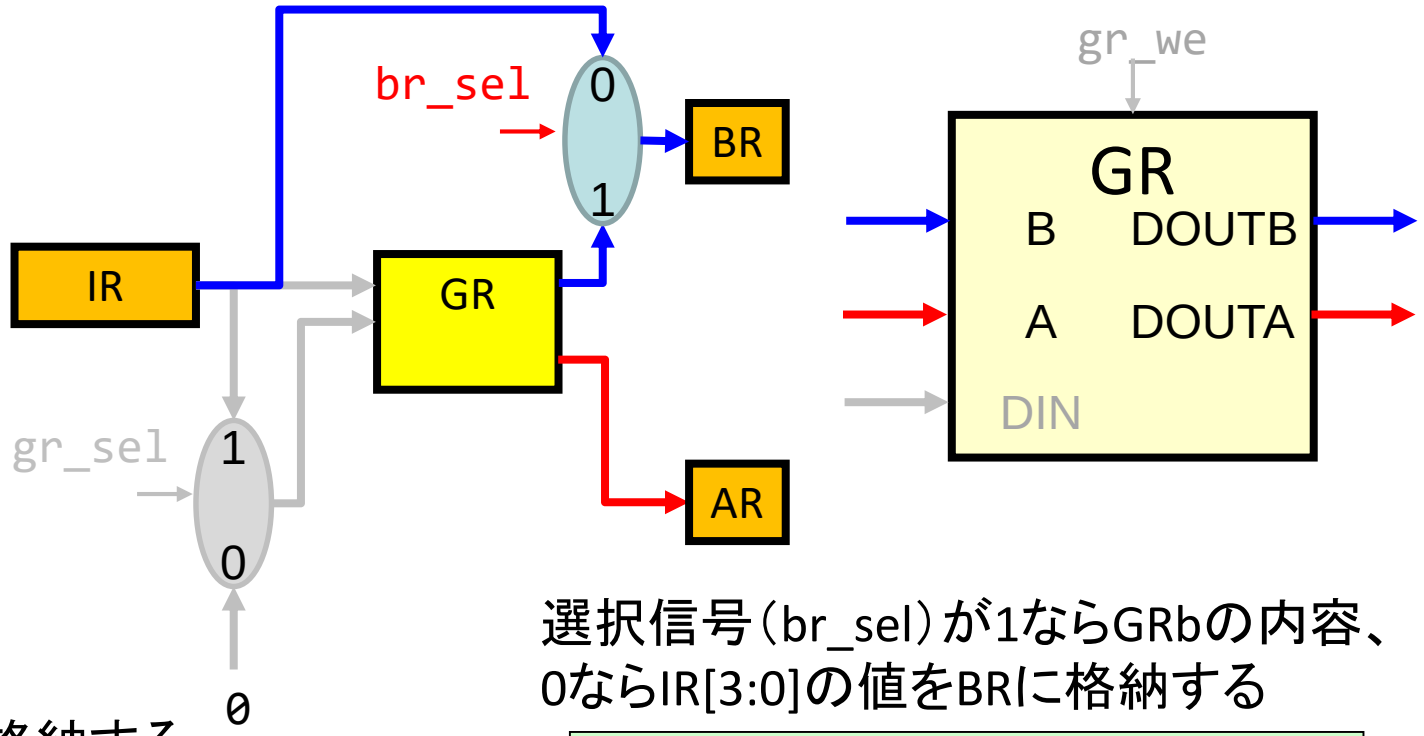




選択信号（gr_sel）が
1ならIR[3:2]の値、0なら0を
GRaのレジスタ番号として指定

```
assign gr_a = (gr_sel == 1‘b1) ? ir_dout[3:2] : 2'b00;
assign gr_b = ir_dout[1:0];
regf gr(.clk(clk), .we(gr_we), .din(gr_din),
        .a(gr_a), .dout_a(gr_dout_a),
        .b(gr_b), .dout_b(gr_dout_b));
```

レジスタ・フェッチ： Register Fetch (RF)



選択信号（br_sel）が1ならGRbの内容、
0ならIR[3:0]の値をBRに格納する

```
assign br_din = (br_sel == 1'b1)
     ? gr_dout_b: ir_dout[3:0];
reg4 br(.clk(clk),
        .rst(1'b1), .we(1'b1),
        .din(br_din),
        .dout(br_dout));
```

GRaの内容をARに格納する

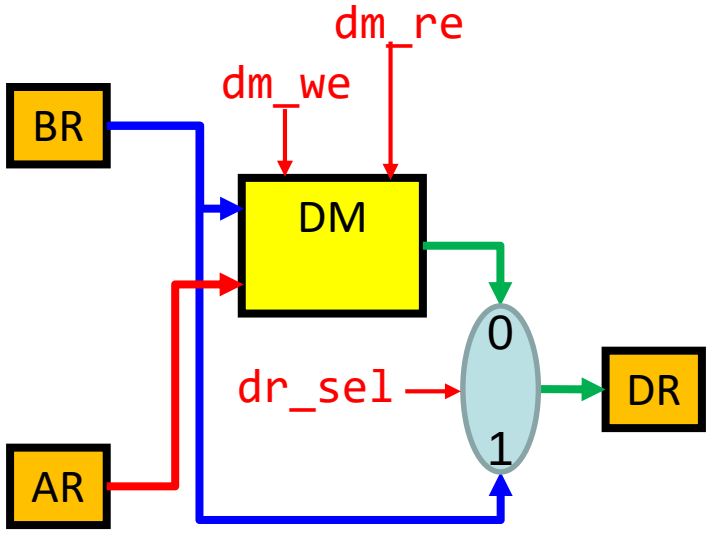
```
assign ar_din = gr_dout_a;
reg4 ar(.clk(clk),
        .rst(1'b1), .we(1'b1),
        .din(ar_din),
        .dout(ar_dout));
```

命令実行：Execution (EX)

アドレス（dm_adrs）によって
読み出し先や書き込み先を指定

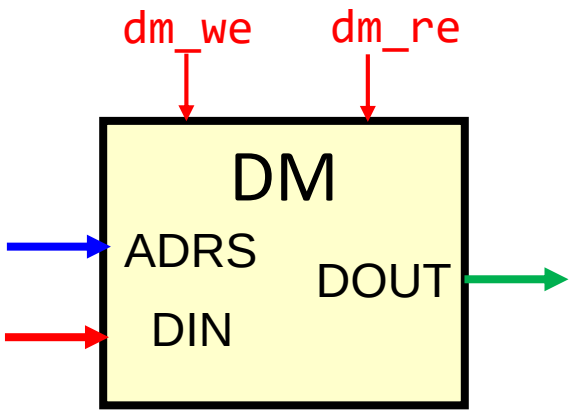
```
assign dm_adrs = br_dout;
assign dm_din = ar_dout;
ram dm(.clk(clk), .re(dm_re), .we(dm_we),
       .adrs(dm_adrs),
       .din(dm_din), .dout(dm_dout));
```

```
assign dr_din = (dr_sel == 1'b1)
     ? br_dout : dm_dout;
reg4 dr(.clk(clk), .rst(1'b1), .we(1'b1),
        .din(dr_din), .dout(dr_dout));
```



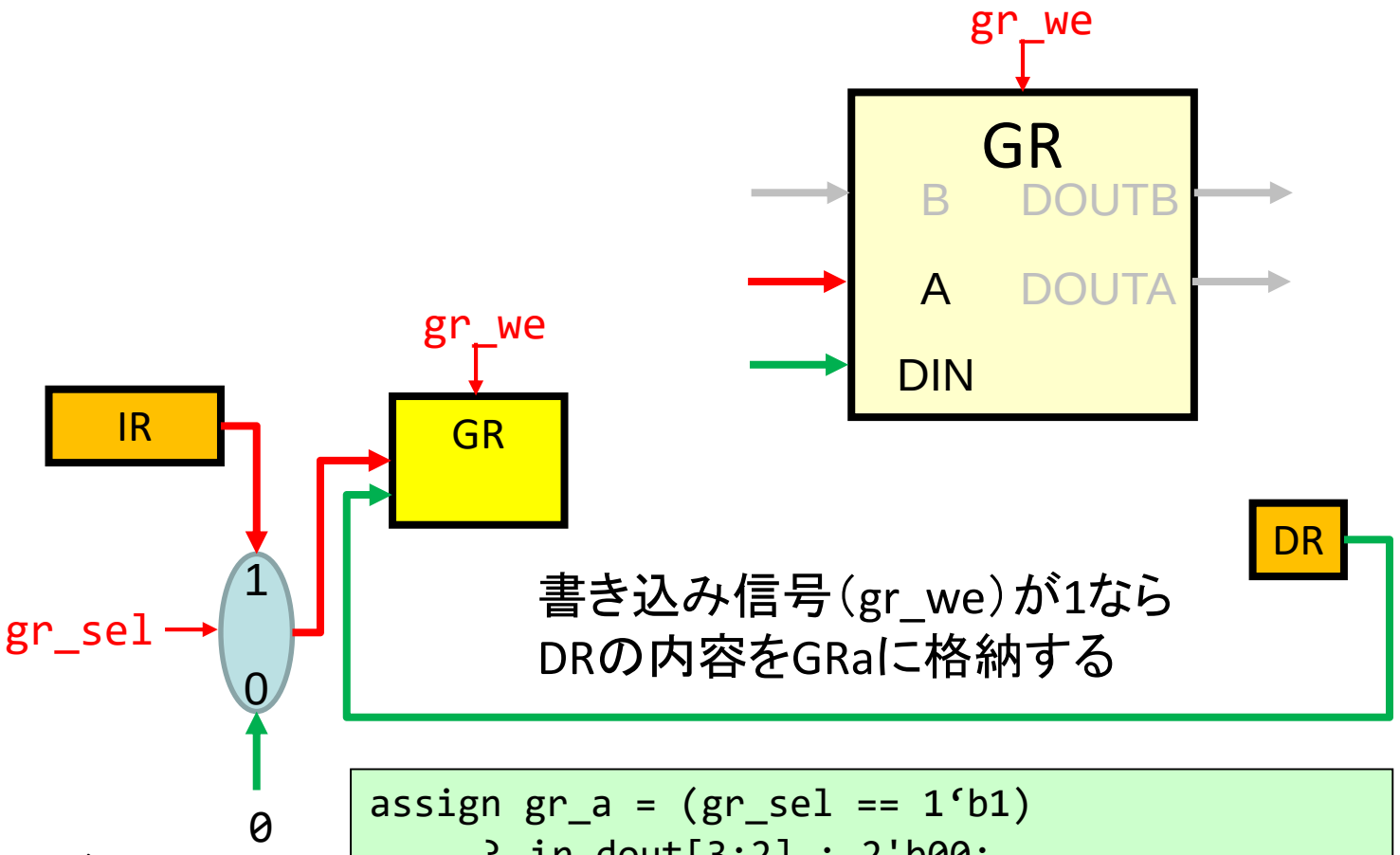
読み出し信号（dm_re）が1なら
DMの内容（データ）を読み出し

書き込み信号（dm_we）が1なら
データ（dm_din）をDMに書き込み



選択信号（dr_sel）が
1ならBRの内容、
0ならDMの内容を
選択してDRに格納

書き込み：Write Back (WB)

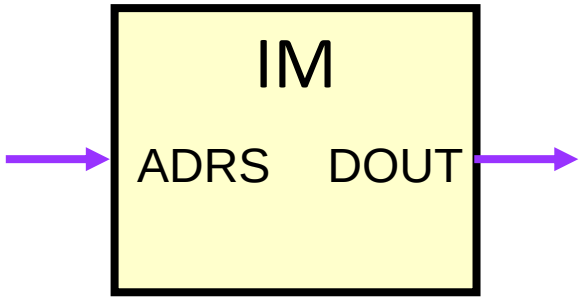


選択信号（gr_sel）が
1ならIR[3:2]の値、0なら0を
GRaのレジスタ番号として指定

```
assign gr_a = (gr_sel == 1‘b1)
     ? ir_dout[3:2] : 2'b00;
assign gr_din = dr_dout;
regf gr(.clk(clk), .we(gr_we), .din(gr_din),
        .a(gr_a), .dout_a(gr_dout_a),
        .b(gr_b), .dout_b(gr_dout_b));
```

命令メモリ(IM)

```
module rom(
  adrs, dout
);
  input  [3:0] adrs;
  output [7:0] dout;
  reg    [7:0] dout;
  always@(adrs) begin
    case(adrs)
      4'b0000: dout <= 8'h06; /* LDI 6 */
      ...
      default: dout <= 8'hxx;
    endcase
  end
endmodule
```

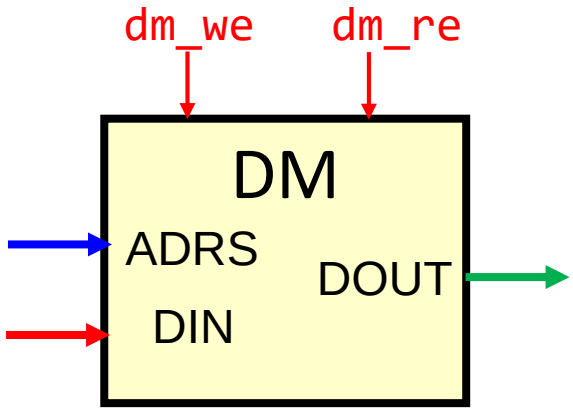


指定したアドレス(adrs)の内容(命令)を読み出し

データメモリ (DM)

```
module ram(
  clk, re, we, adrs, din, dout
);
  input         clk, re, we;
  input  [3:0] adrs, din;
  output [3:0] dout;
  reg    [3:0] data [0:15];
  always@(posedge clk) begin
    if(we == 1'b1) begin
      data[adrs] <= din;
    end
  end
  assign dout = (re == 1'b1)
       ? data[adrs] : 4'bxxxx;
endmodule
```

16個の
4bitレジスタ



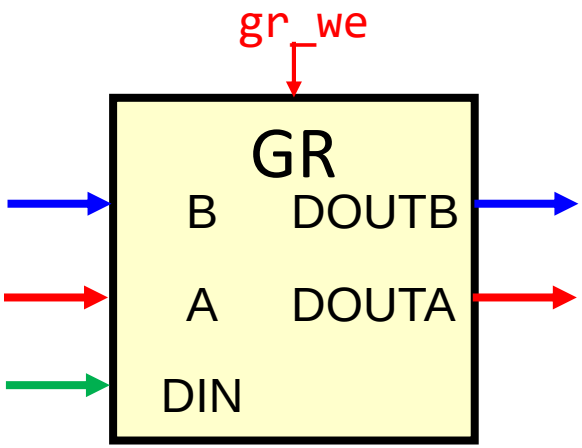
書き込み信号（we）が1なら
指定したアドレス（adrs）にデータ（din）を書き込み

読み出し信号（re）が1なら
指定したアドレス（adrs）のデータを読み出し

汎用レジスタ(GR)

```
module regf(
  clk, we, din, a, dout_a, b, dout_b
);
  input         clk, we;
  input  [3:0] din;
  input  [1:0] a, b;
  output [3:0] dout_a, dout_b;
  reg    [3:0] data [0:3];
  always@(posedge clk) begin
    if(we == 1'b1) begin
      data[a] <= din;
    end
  end
  assign dout_a = data[a];
  assign dout_b = data[b];
endmodule
```

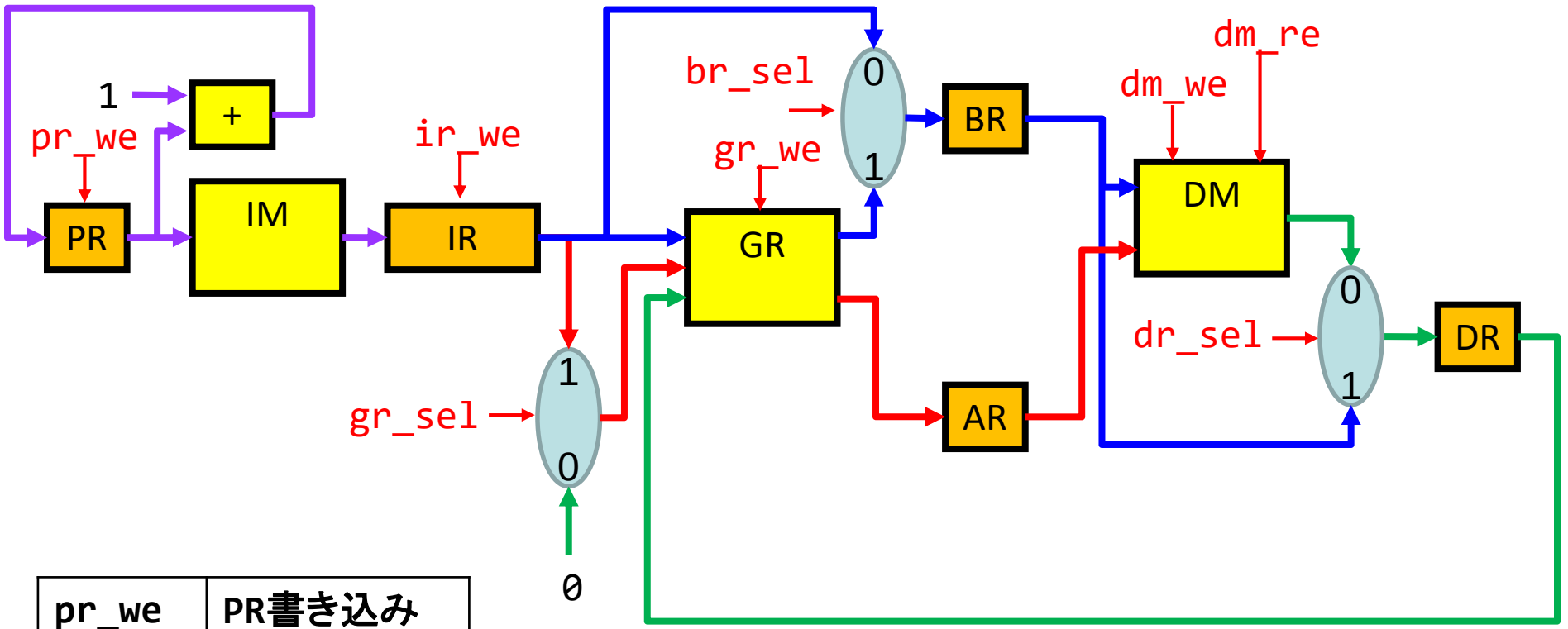
4個の
4bitレジスタ



書き込み信号(we)が1なら
指定した番号(a)のレジスタにデータ(din)を書き込み

指定した番号(a, b)のレジスタの内容をそれぞれ読み出し

CPUの制御信号



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

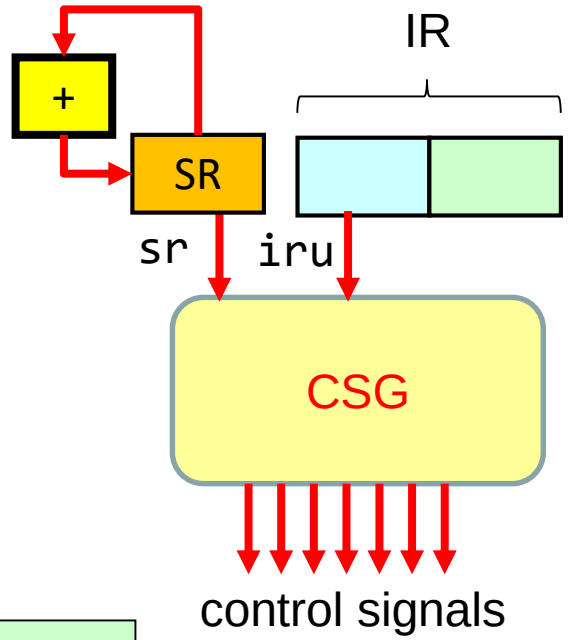
gr_sel	IR[3:2] または 0
br_sel	GRb　または IR[3:0]
dr_sel	BR　または DM

各命令のRTL動作と制御信号

時刻	LDI (0000)	MOV (0001)	LD (0100)	ST (0101)
0	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_we = 1
1	AR ← xx BR ← IMM br_sel = 0	AR ← xx BR ← GRb br_sel = 1	AR ← xx BR ← GRb br_sel = 1	AR ← GRa BR ← GRb gr_sel = 1 br_sel = 1
2	DR ← BR dr_sel = 1	DR ← BR dr_sel = 1	DR ← DM[BR] dm_re = 1 dr_sel = 0	DM[BR] ← AR dm_we = 1
3	GR0 ← DR gr_sel = 0 gr_we = 1	GRa ← DR gr_sel = 1 gr_we = 1	GRa ← DR gr_sel = 1 gr_we = 1	

CPUの制御回路

```
reg8 ir(.clk(clk), .rst(rst), .we(ir_we),
        .din(ir_din), .dout(ir_dout));
```



SRに1を加算
2bitなので3(11)の次は0(00)

```
assign sr_din = sr_dout + 2'b01;
reg2 sr(.clk(clk), .rst(rst),
        .din(sr_din), .dout(sr_dout));
```

```
csg csg(
  .iru(ir_dout[7:4]), .sr(sr_dout),
  .gr_sel(gr_sel), .br_sel(br_sel), .dr_sel(dr_sel),
  .ir_we(ir_we), .pr_we(pr_we), .gr_we(gr_we),
  .dm_re(dm_re), .dm_we(dm_we));
```

操作コードIR[7:4]の内容と
ステートレジスタSRの内容に応じて
各制御信号を出力

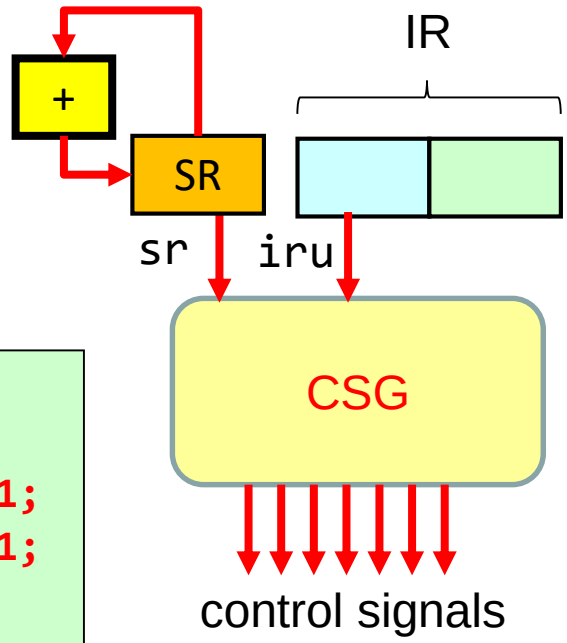
IR：命令レジスタ
SR：ステートレジスタ
CSG：制御信号生成回路

制御信号生成回路 (CSG)

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

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

```
case(sr)
  2'b00: begin
    ir_we <= 1'b1;
    pr_we <= 1'b1;
  end
  2'b01: begin
    case(iru)
      ...
    endcase
    ...
  end
  ...
endcase
```



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

制御信号のデフォルト値を指定すれば以降の記述は単純
IRとSRに対する制御信号の動作を記述

課題

（1）配布した初期バージョンのCPUは
以下の4つの命令が実行できる。

LDI	即値ロード
MOV	レジスタ間転送
LD	ロード
ST	ストア

命令メモリの内容を変更し、
右のテストプログラムを実行せよ。

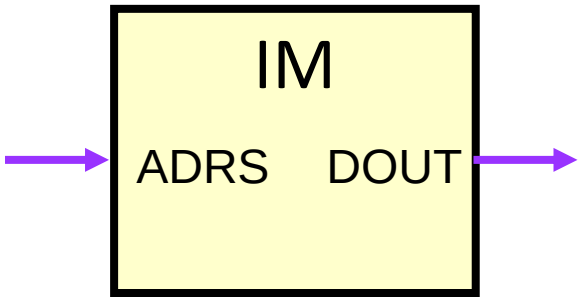
● テストプログラム

```
LDI   5
MOV   GR1, GR0
LDI   2
ST    GR1, GR0
LD    GR2, GR0
MOV   GR3, GR2
```

命令メモリ (IM)

命令メモリの各アドレスに対して命令の機械語を記述せよ。

```
module rom(
  adrs, dout
);
  input  [3:0] adrs;
  output [7:0] dout;
  reg    [7:0] dout;
  always@(adrs) begin
    case(adrs)
      4'b0000: dout <= 8'hxx;
      4'b0001: dout <= 8'hxx;
      ...
      default: dout <= 8'hxx;
    endcase
  end
endmodule
```



機械語の定数を16進数（8’hxx）ではなく、
2進数（8’bxxxxxxxx）で記述しても良い。
また、以下の4つの記述はいずれも
8ビットの定数 01010110 を表しており、
どの記述も同じである。

① 8’h56
② {4’h5,　2’h1,　2’h2}
③ {4’b0101, 2’b01, 2’b10}
④ 8’b01010110

（ST GR1, GR2 の機械語）

課題

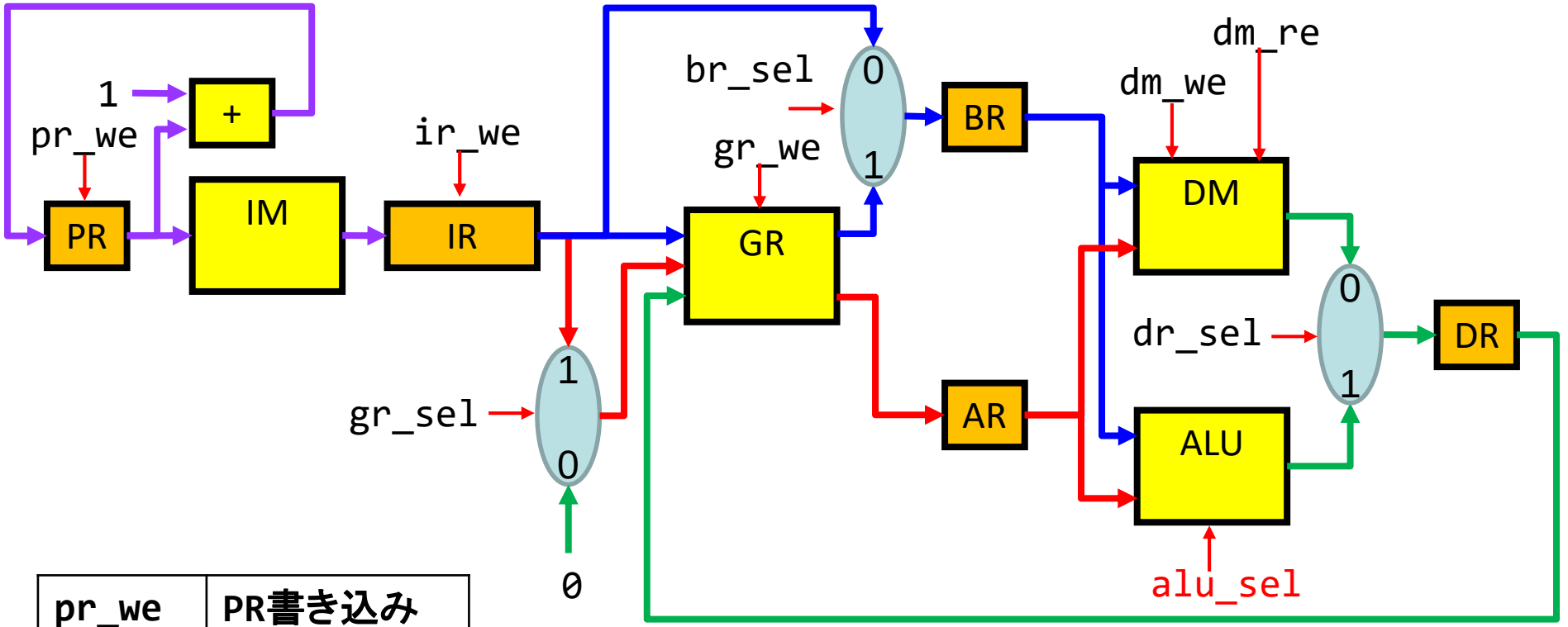
（2）以下の命令を実行できるようにCPUに変更を加えよ。

ADD　加算
SUB　減算

時刻	LDI (0000)	MOV (0001)	ADD (0010)	SUB (0011)
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 ← xx BR ← IMM	AR ← xx BR ← GRb	AR ← GRa BR ← GRb	AR ← GRa BR ← GRb
2	DR ← BR	DR ← BR	DR ← AR + BR	DR ← AR – BR
3	GR0 ← DR	GRa ← DR	GRa ← DR	GRa ← DR

演算回路の追加

ALUを追加
2bitの選択信号(alu_sel)を追加



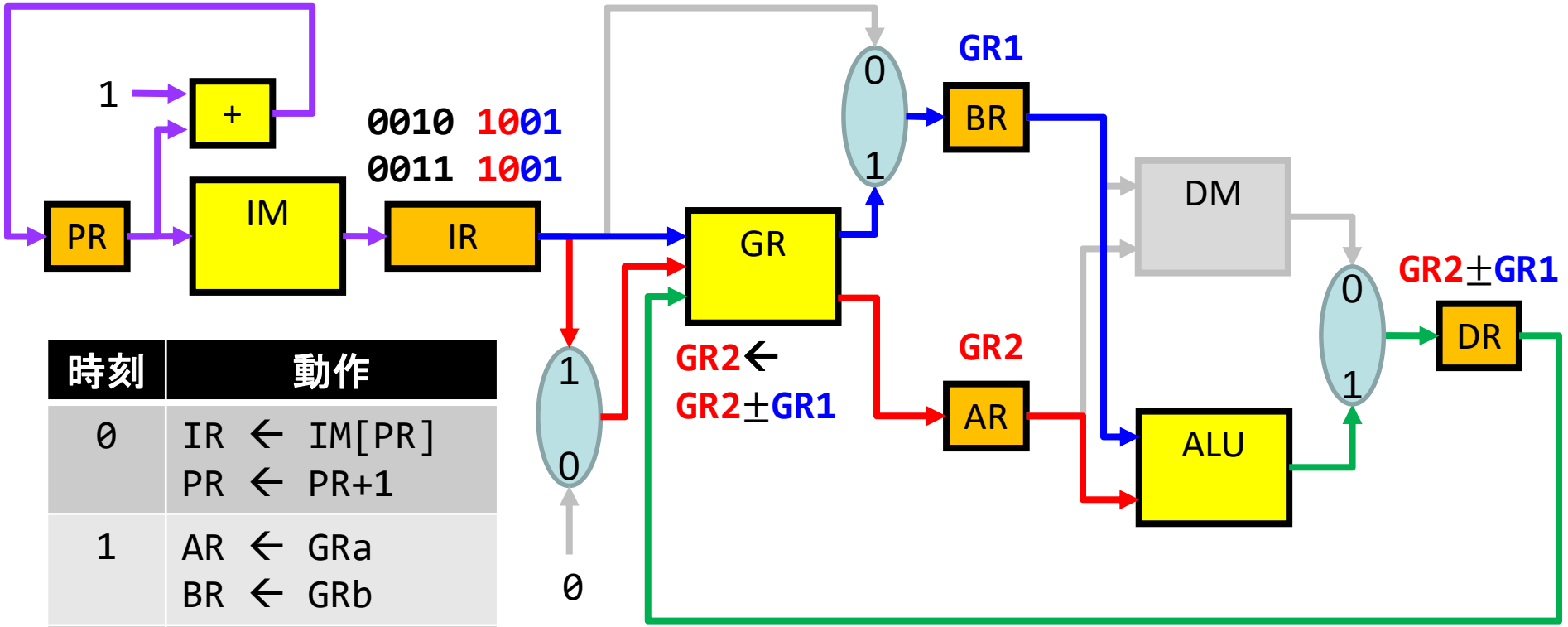
pr_we	PR書き込み
ir_we	IR書き込み
gr_we	GR書き込み
dm_we	DM書き込み
dm_re	DM読み出し

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

ADD,SUB命令の動作

ADD GR2,GR1	0010	1001	GR2 ← GR2 + GR1
SUB GR2,GR1	0011	1001	GR2 ← GR2 - GR1

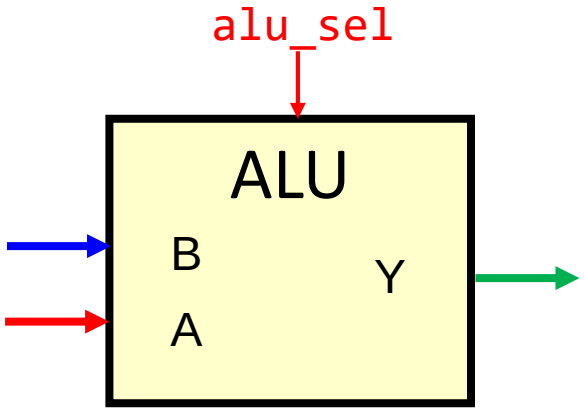


時刻	動作
0	IR ← IM[PR] PR ← PR+1
1	AR ← GRa BR ← GRb
2	DR ← AR±BR
3	GRa ← DR

算術論理演算ユニット(ALU)

```
module alu(
  a, b, sel, y
);
  input  [3:0] a, b;
  input  [1:0] sel;
  output [3:0] y;
  reg    [3:0] y;

  always@(a or b or sel) begin
    case(sel)
      ...
      ...
      ...
      default: ...;
    endcase
  end
endmodule
```



2bitの選択信号(alu_sel)に対して
ALUの出力を選択

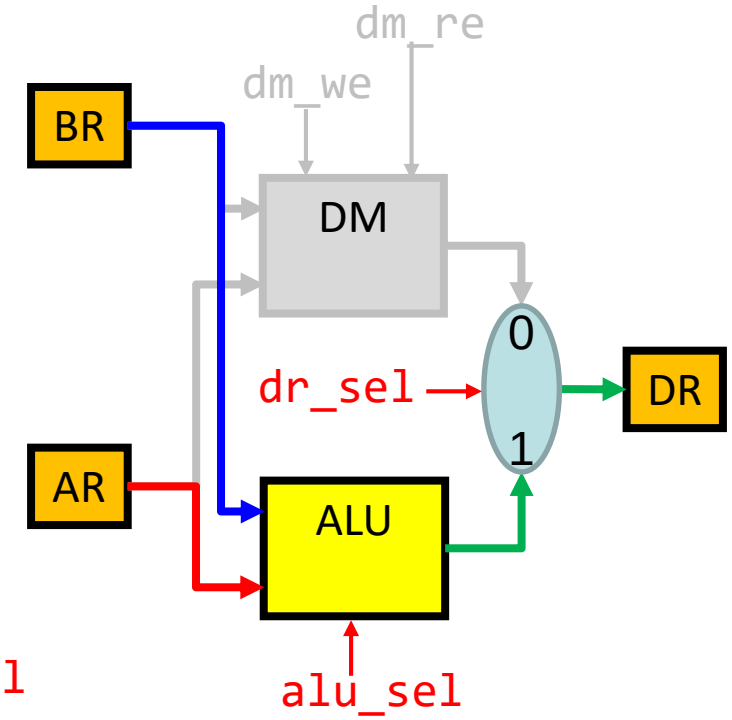
alu_sel	1	B
	2	A+B
	3	A-B

ALUの追加

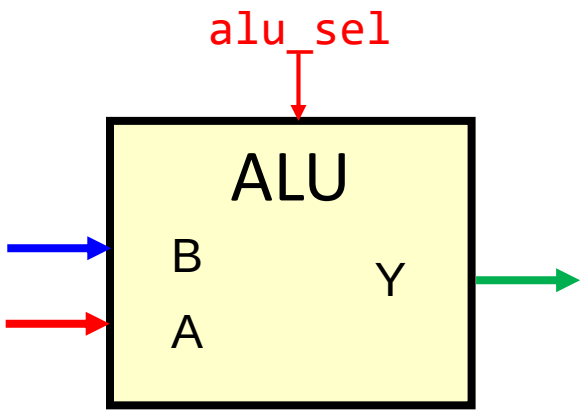
ARとBRの内容を演算し
選択信号(alu_sel)で出力を選択

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

```
assign dr_din = (dr_sel == 1'b1)
     ? alu_dout : dm_dout;
reg4 dr(.clk(clk), .rst(1'b1), .we(1'b1),
        .din(dr_din), .dout(dr_dout));
```



選択信号(dr_sel)が1なら
ALUの演算結果をDRに格納



alu_sel	1	B
	2	A+B
	3	A-B

各命令のRTL動作と制御信号

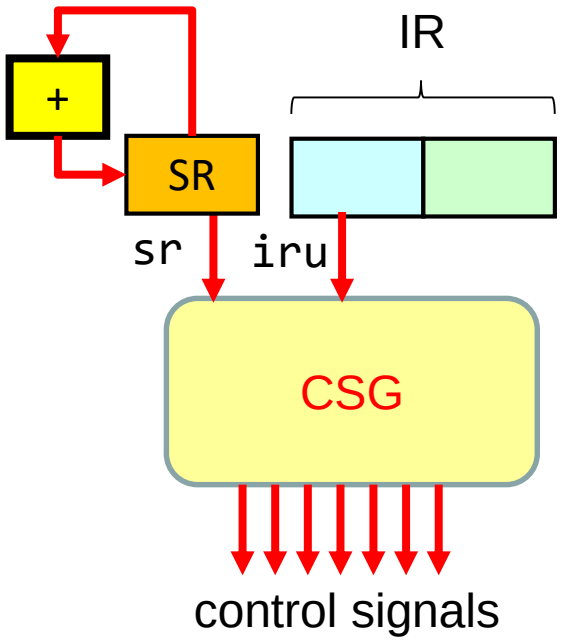
時刻	LDI (0000)	MOV (0001)	ADD (0010)	SUB (0011)
0	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_we = 1	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_we = 1
1	AR ← xx BR ← IMM br_sel = 0	AR ← xx BR ← GRb br_sel = 1	AR ← GRa BR ← GRb gr_sel = 1 br_sel = 1	AR ← GRa BR ← GRb gr_sel = 1 br_sel = 1
2	DR ← BR alu_sel = 01 dr_sel = 1	DR ← BR alu_sel = 01 dr_sel = 1	DR ← AR + BR alu_sel = 10 dr_sel = 1	DR ← AR – BR alu_sel = 11 dr_sel = 1
3	GR0 ← DR gr_sel = 0 gr_we = 1	GRa ← DR gr_sel = 1 gr_we = 1	GRa ← DR gr_sel = 1 gr_we = 1	GRa ← DR gr_sel = 1 gr_we = 1

LD, ST命令はALUを使用しないので制御は変更なし

ALU選択信号の追加

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

```
always@(iru or sr) begin
  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;
  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

2bitの選択信号（alu_sel）を追加
LDI, MOVの制御を修正
ADDとSUBの制御を追加

課題

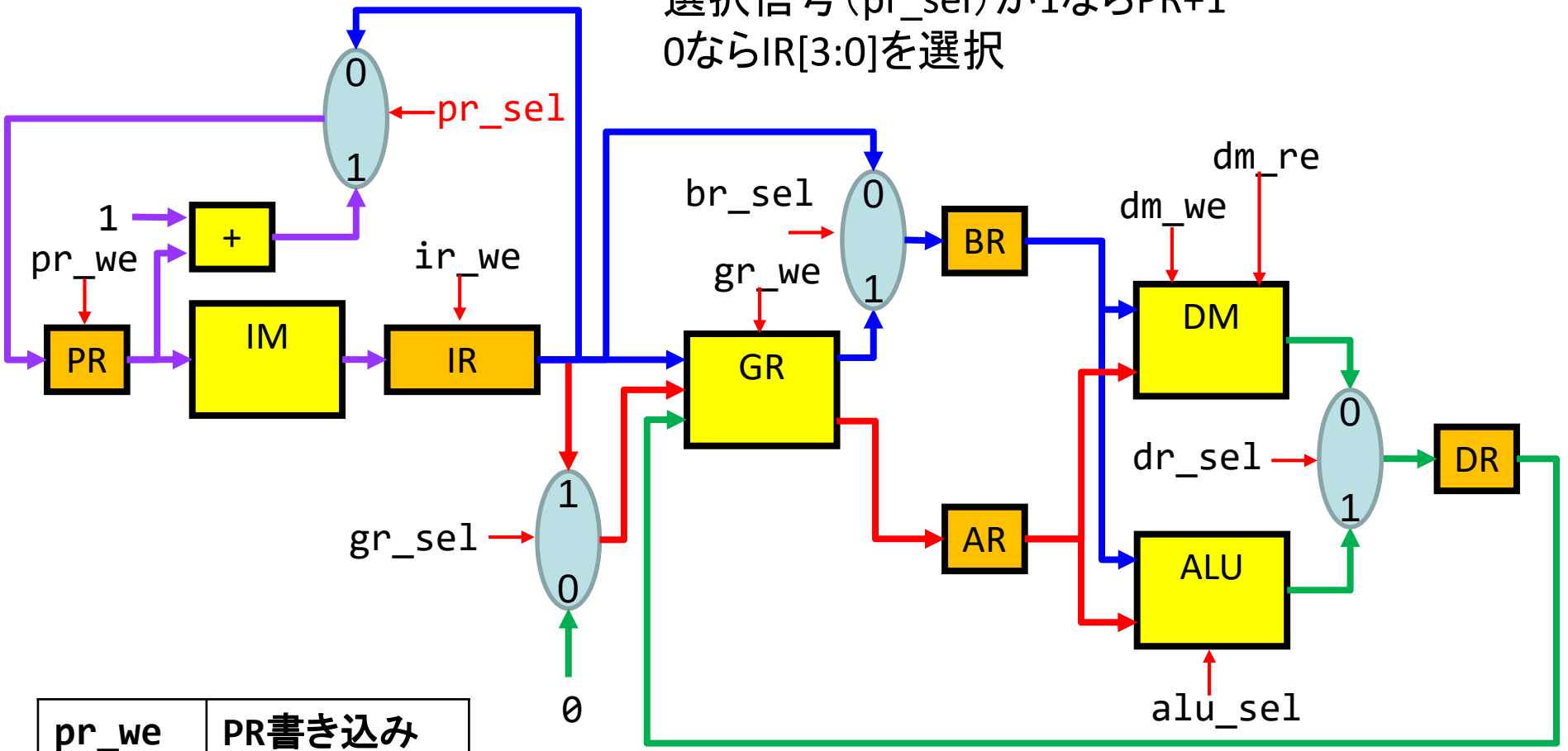
（3）以下の命令を実行できるようにCPUに変更を加えよ。

JMP　　無条件ジャンプ

時刻	JMP (1010)
0	IR ← IM[PR] PR ← PR+1
1	PR ← ADRS

PR選択回路の追加

選択信号（pr_sel）が1ならPR+1
0ならIR[3:0]を選択



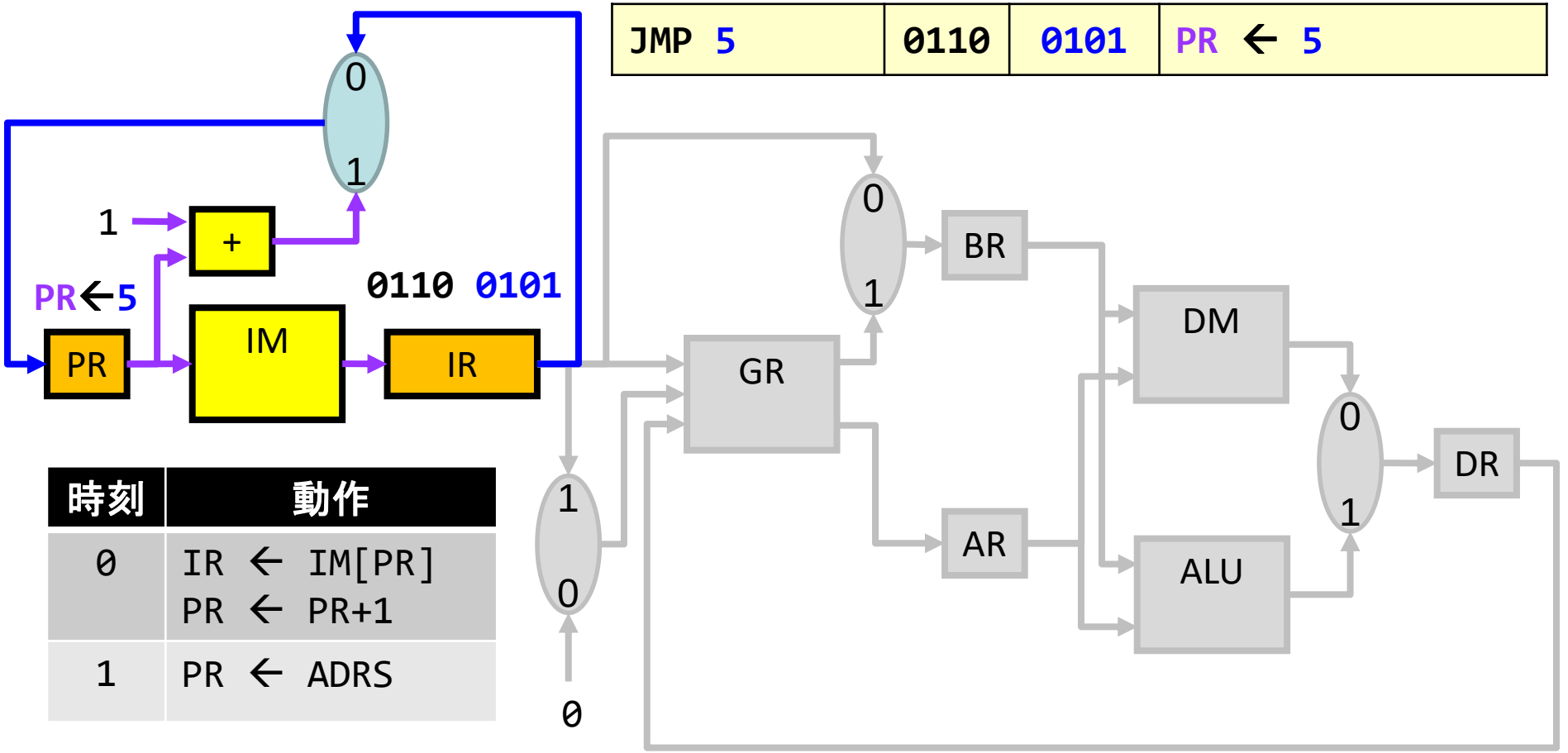
pr_we	PR書き込み
ir_we	IR書き込み
gr_we	GR書き込み
dm_we	DM書き込み
dm_re	DM読み出し

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

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

各命令のRTL動作と制御信号

時刻	JMP (0110)
0	IR ← IM[PR] PR ← PR+1 ir_we = 1 pr_sel = 1 pr_we = 1
1	PR ← ADRS pr_sel = 0 pr_we = 1

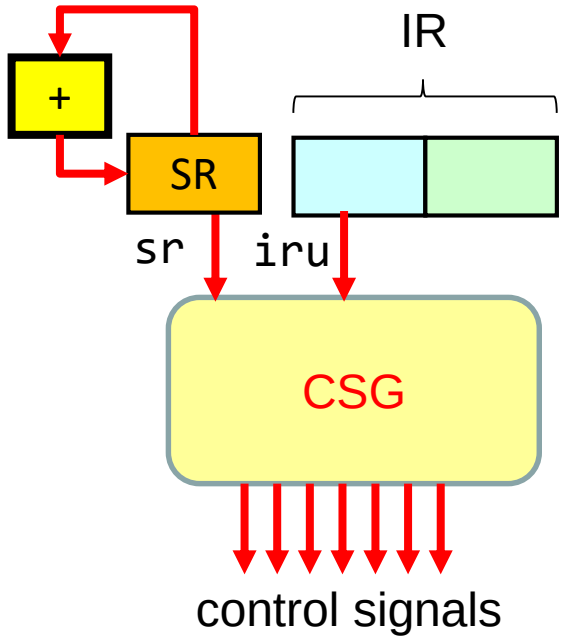
時刻0は命令フェッチの動作
➔ 何の命令か確定していない段階
➔ ジャンプ命令も一旦はPRを繰り上げる

時刻0での pr_sel の動作は全命令で共通
➔ PRを繰り上げるために必要

PR選択信号の追加

```
module csg(
  iru, sr,
  pr_sel, gr_sel, br_sel, dr_sel,
  alu_sel,
  ir_we, pr_we, gr_we,
  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;
  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

選択信号（pr_sel）を追加
全命令共通の制御を修正
JMPの制御を追加

課題

（1）命令メモリを変更し、テストプログラムを実行せよ。

（2）以下の命令を実行できるようにCPUに変更を加えよ。

ADD　加算
SUB　減算

（3）以下の命令を実行できるようにCPUに変更を加えよ。

JMP　無条件ジャンプ

テストプログラム例

課題(1)

```
LDI   5
MOV   GR1, GR0
LDI   2
ST    GR1, GR0
LD    GR2, GR0
MOV   GR3, GR2
```

課題(2)

```
LDI   5
MOV   GR1, GR0
LDI   2
ST    GR1, GR0
LD    GR2, GR0
MOV   GR3, GR2
ADD   GR1, GR0
SUB   GR2, GR0
```

課題(3)

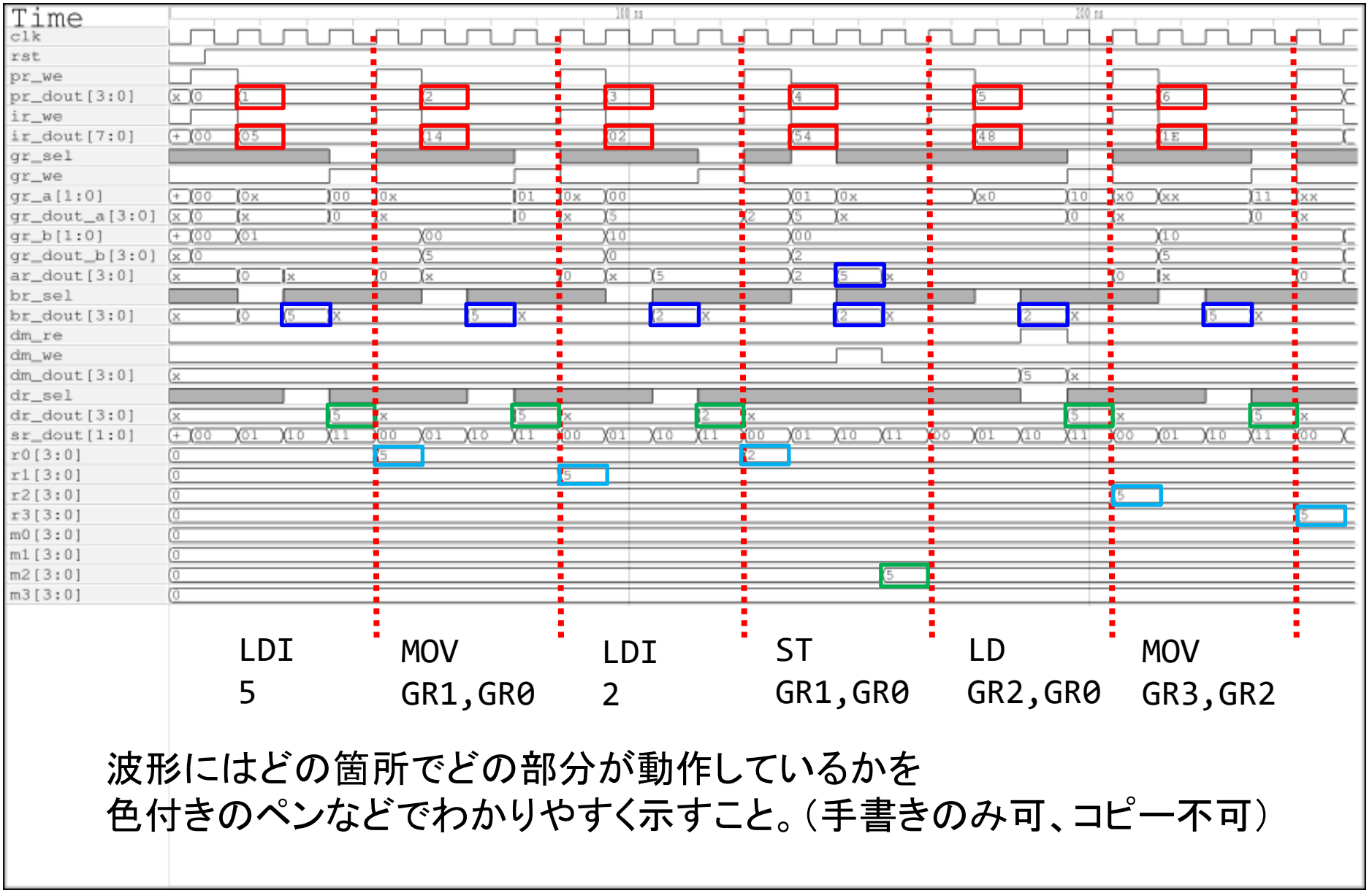
```
LDI   5
MOV   GR1, GR0
LDI   2
ST    GR1, GR0
LD    GR2, GR0
MOV   GR3, GR2
ADD   GR1, GR0
SUB   GR2, GR0
JMP   0
```

課題(1)は上記のプログラムを実行すること。
課題(2)(3)は他のプログラムでテストしてもよい。
すべての命令が正しく動作することを確認せよ。

変更のポイント

- 課題2：2bitのALUの選択信号（alu_sel）を追加
  - ALUに加算と減算の回路を追加
- 課題3：PRの選択信号（pr_sel）を追加
  - PR＋1またはIRの内容を選択する回路を追加
- 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

波形の提出方法



波形にはどの箇所でどの部分が動作しているかを
色付きのペンなどでわかりやすく示すこと。（手書きのみ可、コピー不可）

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）
　※課題ごとに動作に必要な *.v を全て（他のファイルを含めても可）

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）
　※全ての課題についてまとめた一通のレポート

レポートの内容：
iverilog の実行結果（ワープロ可）
gtkwave の波形（印刷可、ペンの追記は手書きのみ可）
結果に対する説明や考察などを記述（ワープロ可）

レポートの表紙：
第5回ハードウェア記述言語レポート
学籍番号、氏名