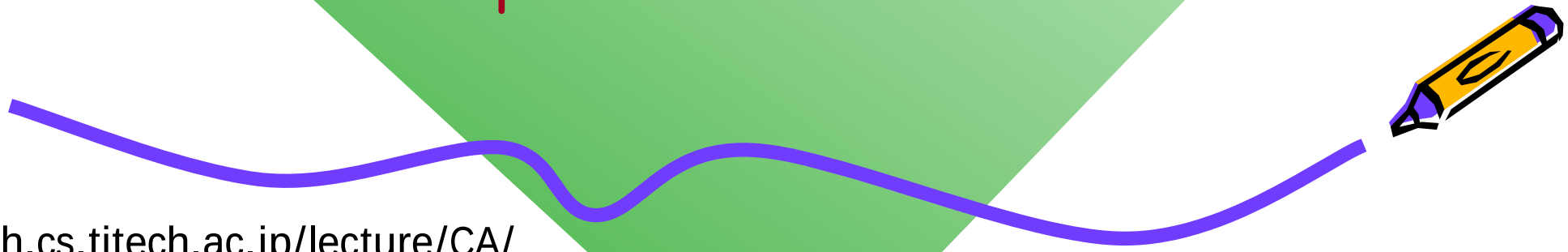
2018年度（平成30年度）版

Ver. 2018-10-11a

Course number: CSC.T363

パイプラインプロセッサ
Computer Architecture

7. パイプラインプロセッサ
Pipelined Processor

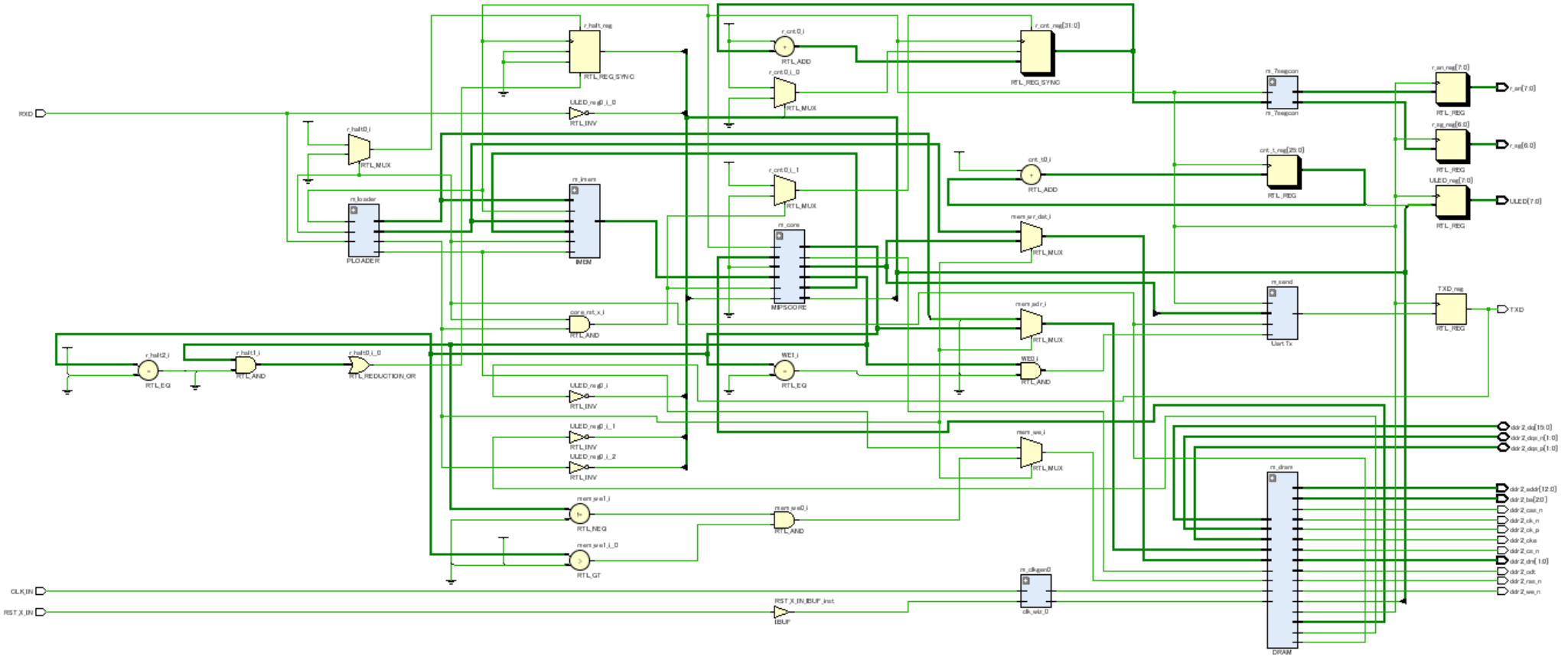


www.arch.cs.titech.ac.jp/lecture/CA/
Room No.W321
Tue 13:20-16:20, Fri 13:20-14:50

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

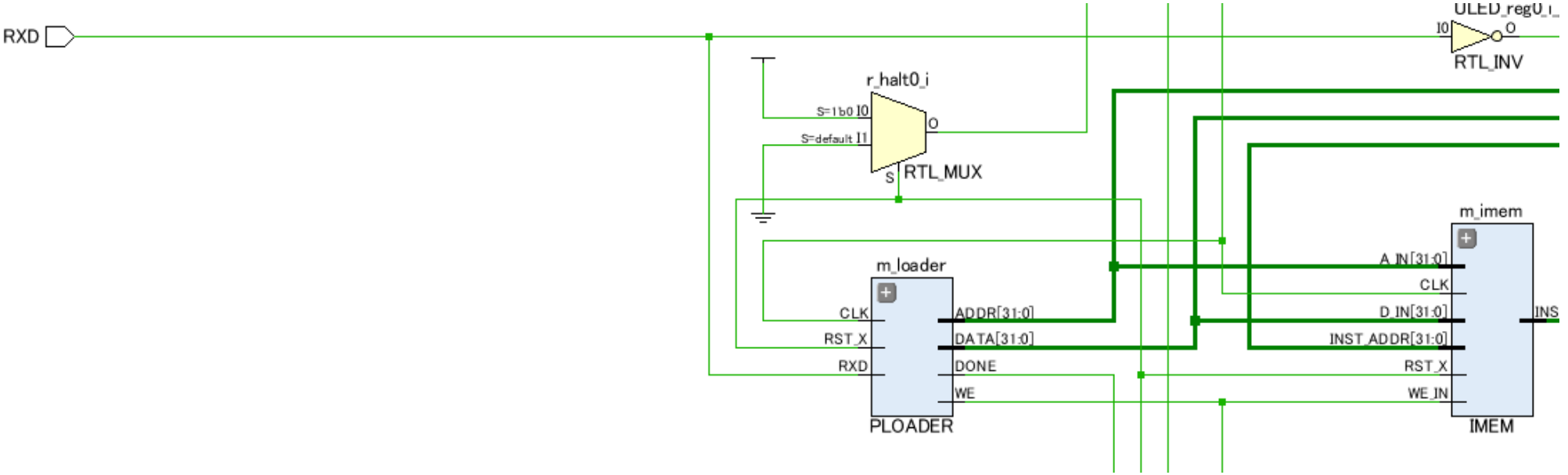
Reference Design (project_24)

- Modules
  - PLOADER, IMEM, MIPSCORE, m_7segcon, UartTx, DRAM, clk_wiz_0



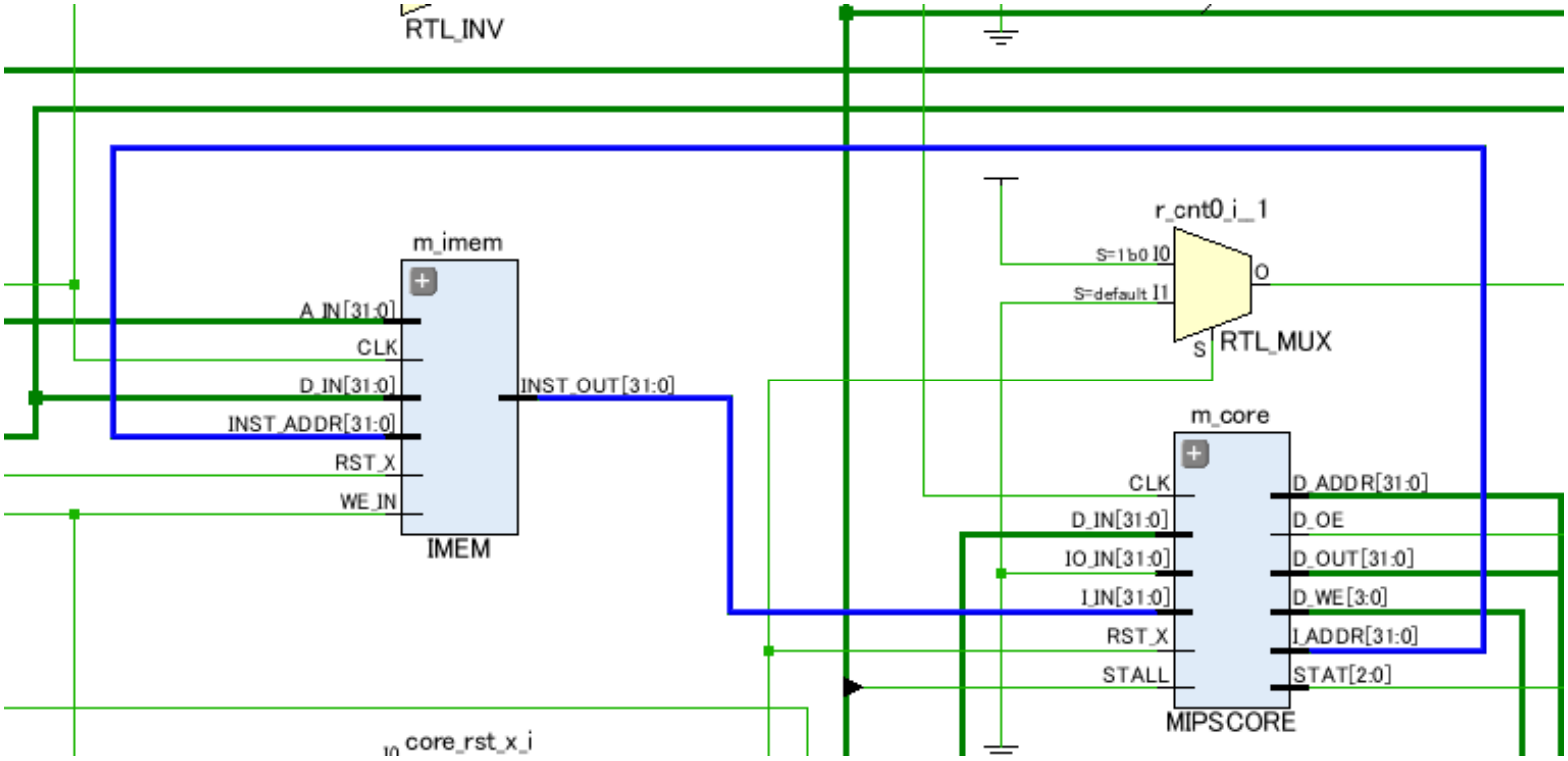
Program loader (PLOADER)

- シリアル通信でRXDからデータを受け取る.
- 32ビットの1つの命令を受け取ると，書き込みアドレスをADDR, 受け取った命令 をDATAを設定し，書き込み信号WE を1にすることで命令メモリIMEM に書き込む.
- 命令メモリに書き込むのは受信する512KBのデータの最初の64KBのみ.



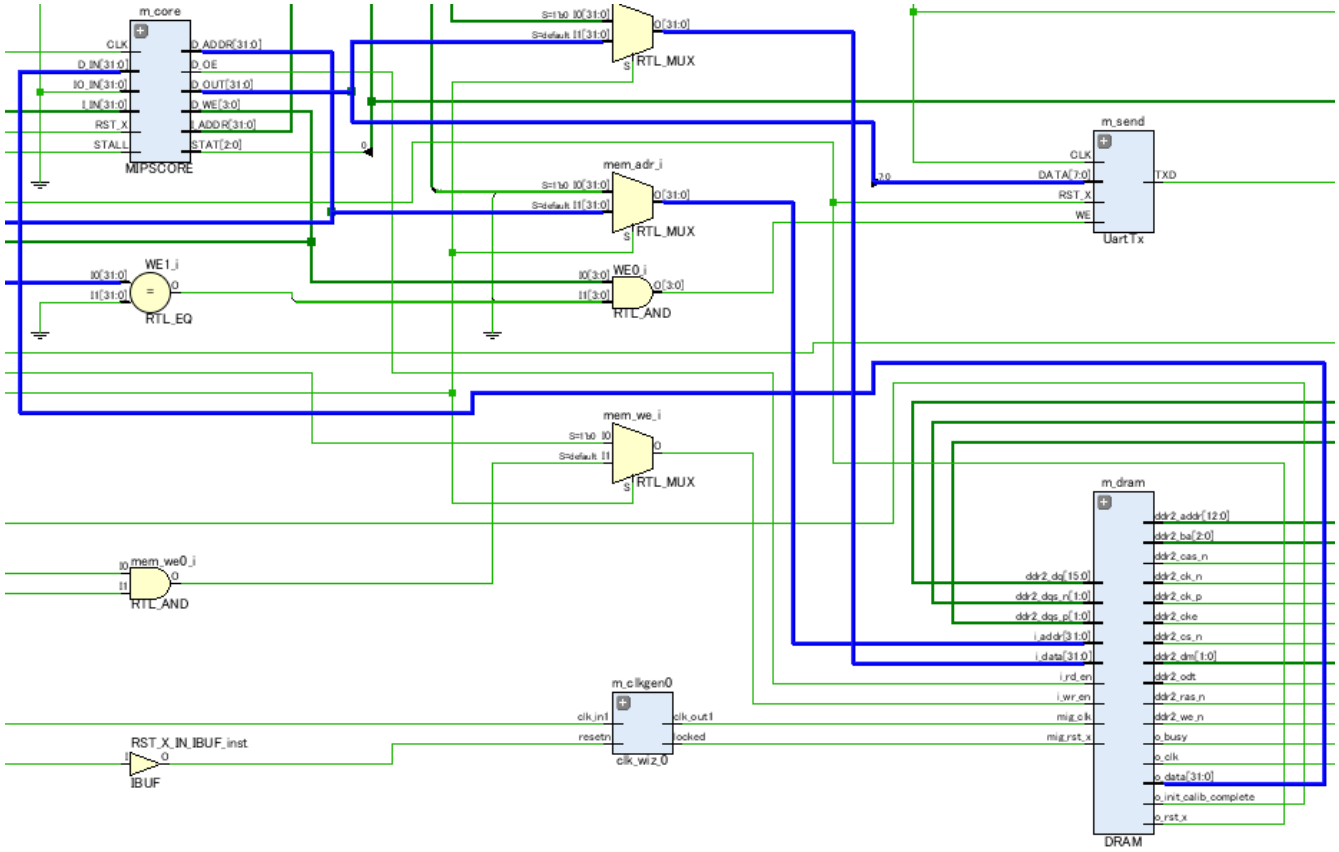
Instruction Memory (IMEM)

- プロセッサ(MIPSCORE)が命令アドレスI_ADDRを出力し，それが命令メモリIMEMの入力INST_ADDRとなる．
- 命令メモリは指定されたアドレスに格納されている命令をINST_OUTに出力して，それがプロセッサの入力I_INとなる．これらが命令フェッチに対応する．



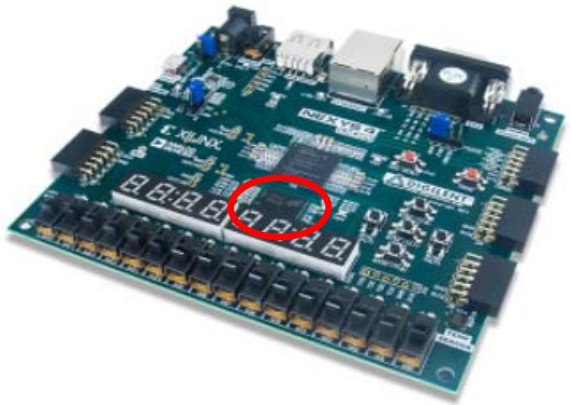
Data Memory (DRAM, m_dram)

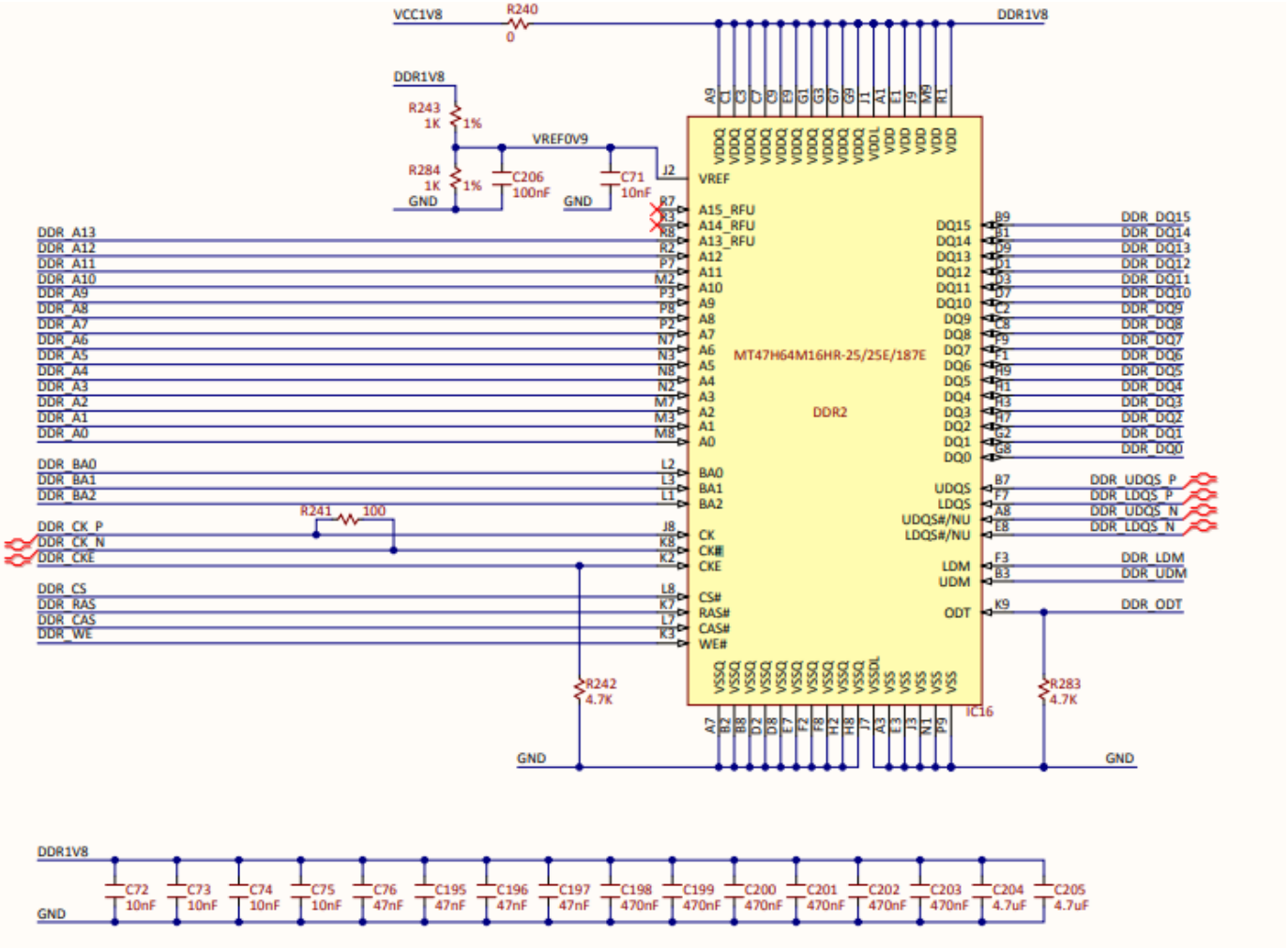
- プロセッサMIPSCOREはストアすべきデータD_OUTを出力し，それがメモリコントローラDRAMの入力i_dataとなる．また，プロセッサはストアすべきデータのアドレスD_ADDRを出力し，それがメモリコントローラの入力i_addrとなる．プロセッサの出力D_WEにより，DRAMへのデータのストアを指示する．
- プロセッサはロードするアドレスD_ADDRを出力し，それがメモリコントローラの入力i_addrとなる．プロセッサの出力D_OEにより，DRAMへのデータのリードを指示する．DRAMから得られたデータがo_dataに出力され，それがプロセッサの入力D_INとなる．



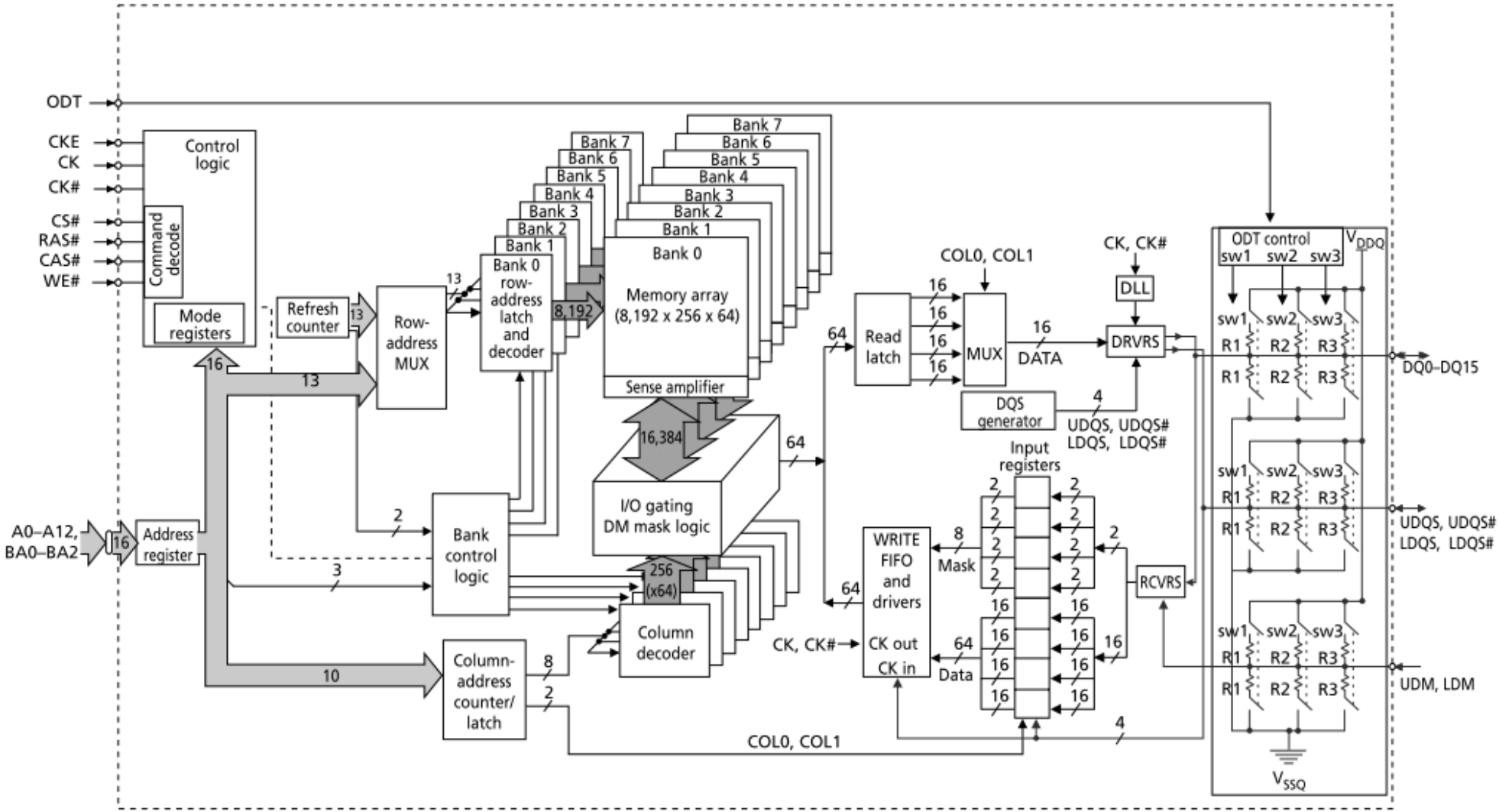
DDR2 Memory on NEXYS 4 DDR

- Micron MT47H64M16HR-25:H DDR2 memory
  - 128MiB DDR2, 16-bit wide interface





Micron MT47H64M16HR-25:H

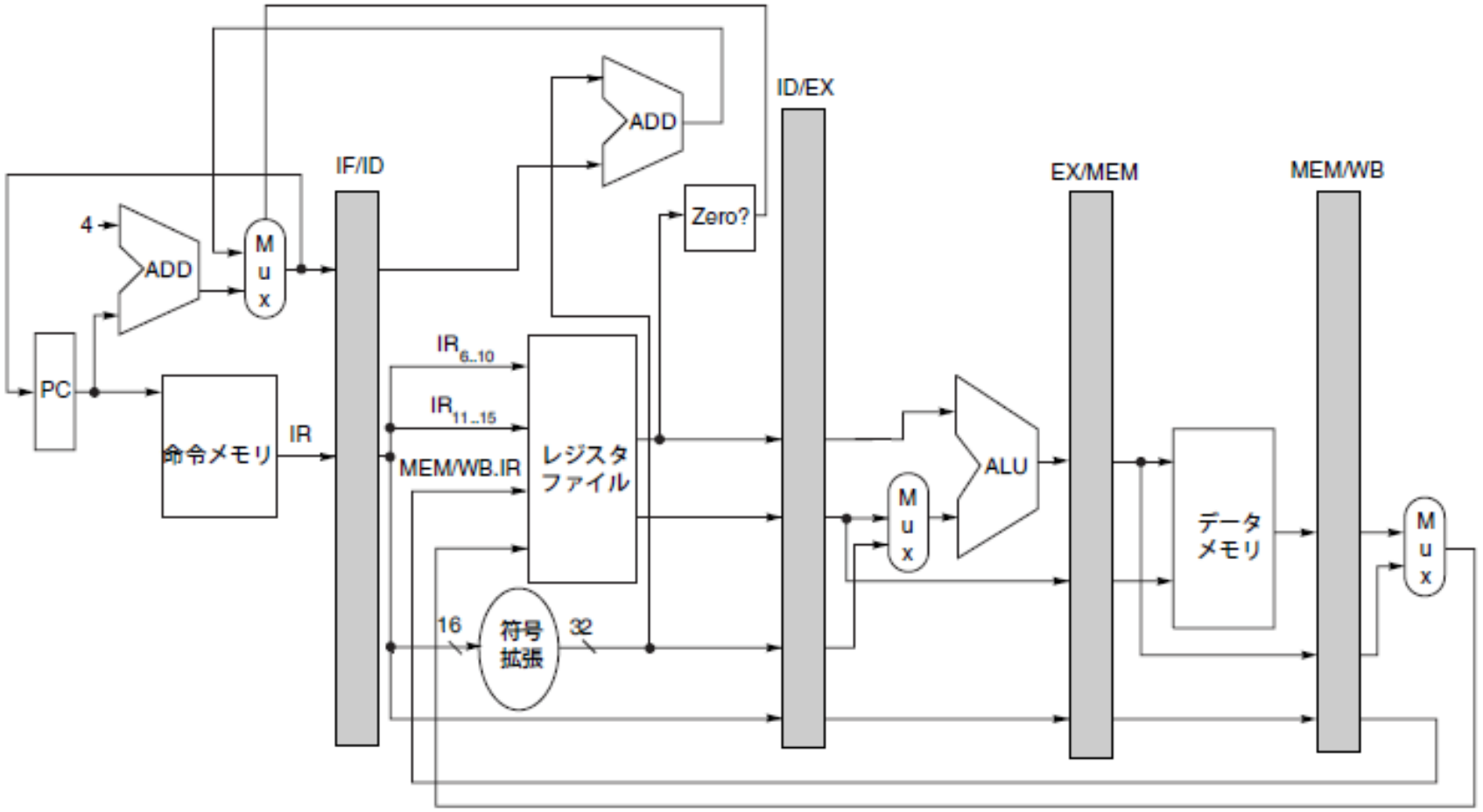


Micron datasheet

プロセッサが命令を処理するための5つのステップの修正

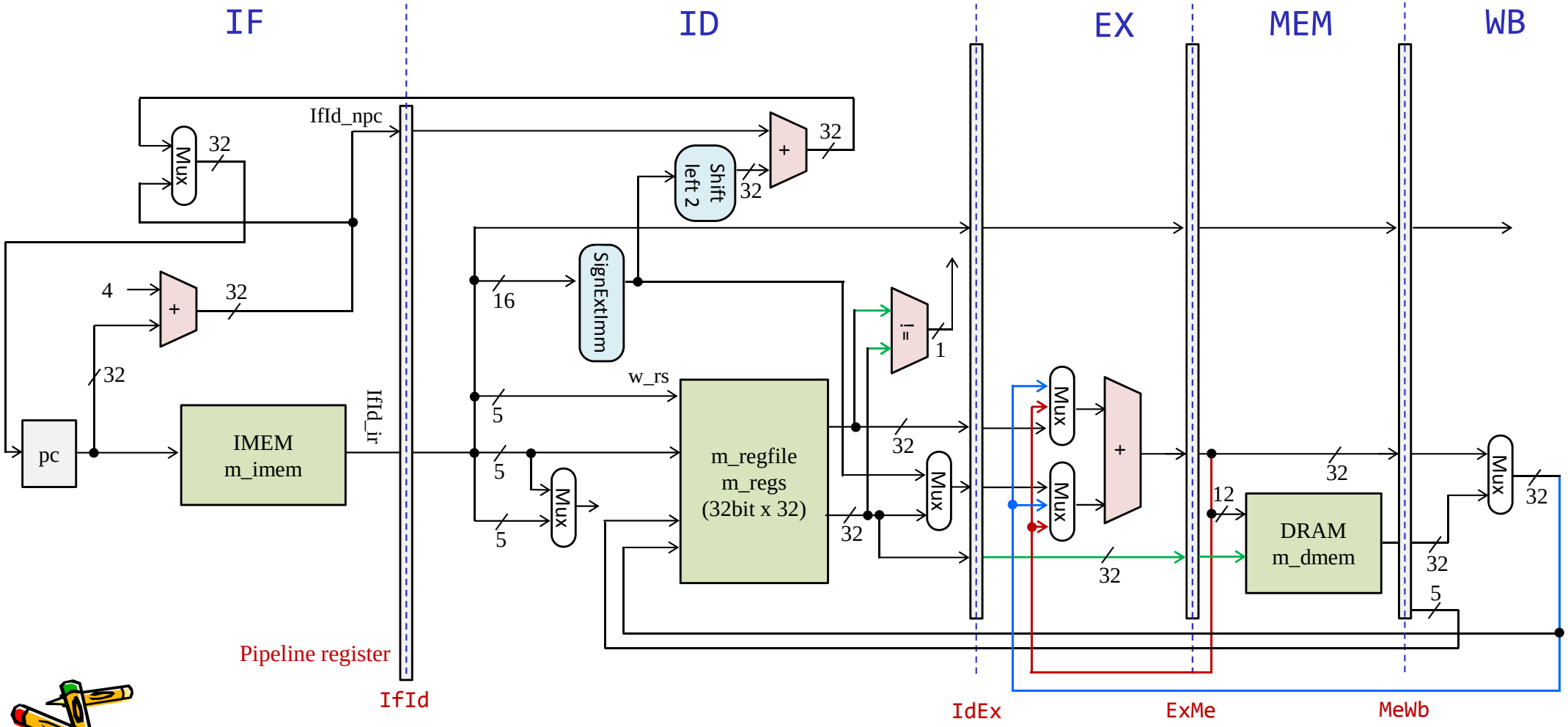
- IF (Instruction Fetch)
  メモリから命令をフェッチする.
- ID (Instruction Decode)
  命令をデコード(解読)しながら, レジスタの値を読み出す.
  分岐命令である可能性を考慮し, 読み出されたレジスタの間で一致比較を行う. 命令のオフセットフィールドを符号拡張し, インクリメントされたPCに符号拡張されたオフセットを足し合わせて分岐先のアドレスを計算する. 条件が成立した場合には分岐先アドレスをPCにセットして, このステージで分岐命令を完了させる.
- EX (Execution)
  命令操作の実行またはアドレスの生成を行う.
- MEM (Memory Access)
  必要であれば, データ・メモリ中のオペランドにアクセスする.
- WB (Write Back)
  必要であれば, 結果をレジスタに書き込む.

プロセッサのデータパス（パイプライン処理）



リファレンスデザインに含まれるプロセッサ

- Operating frequency: 50MHz
- リファレンスデザインには，5段パイプライン処理の典型的なMIPSプロセッサ（のサブセット）が採用されている．下の図からレジスタ名などが変わっているので注意．



プロセッサMIPSCORE: インタフェース

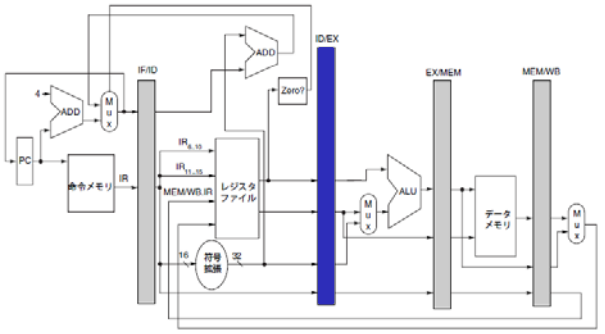
- メモリアクセスステージでロード，ストア命令が処理される時にDRAMにアクセスする．この時，STALL信号が1となり，その間はプロセッサの処理をストールさせる．
- STAT はプロセッサの状態を出力するために使用する．
- I_ADDR, I_IN は命令をフェッチするために使用する．I_ADDR は pc の値．
- D_ADDR, D_IN, D_OUT, D_OE, D_WE はデータメモリのリード，ライトのために使う．
- IO_IN はメモリマップドIOからの入力のために用いるが，このデザインでは使用しない．

```
1  /*************************************************************************************************/
2  /* Sample Verilog HDL Code for CSC.T363 Computer Architecture                  Arch Lab. TOKYO TECH */
3  /*************************************************************************************************/
4  `include "define.v"
5  /*************************************************************************************************/
6  /* MipsCore: 32bit-MIPS 5-stage pipelining processor                                             */
7  /*************************************************************************************************/
8  module MIPSCORE(CLK, RST_X, STALL, STAT, I_ADDR, I_IN, D_ADDR, D_IN, D_OUT, D_OE, D_WE, IO_IN);
9      input  wire         CLK, RST_X, STALL;
10     output reg  [2:0]   STAT;
11     output wire [`ADDR] I_ADDR, D_ADDR;
12     input  wire [31:0]  I_IN, D_IN, IO_IN;
13     output wire [31:0]  D_OUT;
14     output wire [3:0]   D_WE;
15     output wire         D_OE;
16
17     reg [`ADDR]  pc;          // program counter
18     assign I_ADDR = pc;
```

プロセッサMIPSCORE: パイプラインレジスタ

- IfId_ から始まるレジスタは命令フェッチでデコードの間のパイプラインレジスタ

```
19    /*****************************************************************************************/
20    reg [`ADDR] IfId_npc;     // IF-ID pipeline reg: next pc, pc + 4
21    reg [31:0]  IfId_ir;      // IF-ID pipeline reg: instruction
22
23    reg [`ADDR] IdEx_npc;     // ID-EX pipeline reg: next pc, pc + 4
24    reg [31:0]  IdEx_rrs;     // ID-EX pipeline reg: fetched operand of R[rs]
25    reg [31:0]  IdEx_rrt;     // ID-EX pipeline reg: fetched operand of R[rt]
26    reg [4:0]   IdEx_dst;     // ID-EX pipeline reg: decoded destination reg number
27    reg [31:0]  IdEx_ir;      // ID-EX pipeline reg: instruction
28    reg [`OPNT] IdEx_opn;     // ID-EX pipeline reg: decoded operation ID
29    reg [`ATTR] IdEx_attr;    // ID-EX pipeline reg: decoded instruction attribute
30
31    reg [4:0]   ExMa_dst;     // EX-MA pipeline reg: decoded destination reg number
32    reg [31:0]  ExMa_rslt;    // EX-MA pipeline reg: execution result
33    reg [3:0]   ExMa_mwe;     // EX-MA pipeline reg: mem write enable
34    reg         ExMa_oe;      // EX-MA pipeline reg: data memory output enable
35    reg [4:0]   ExMa_lds;     // EX-MA pipeline reg: load selector
36    reg [31:0]  ExMa_std;     // EX-MA pipeline reg: store data
37
38    reg [4:0]   MaWb_dst;     // MA-WB pipeline reg: decoded destination reg number
39    reg [31:0]  MaWb_rslt;    // MA-WB pipeline reg: execution result
```



Hazards make pipelining hard

- 命令を適切なサイクルで実行できないような状況が存在する．これをハザード(hazard)と呼ぶ．
  - 構造ハザード (structural hazard)
    - オーバラップ実行する命令の組み合わせをハードウェアがサポートしていない場合．資源不足により生じる．
  - データ・ハザード(data hazard)
    - データの受け渡しの制約によって生じるハザード
  - 制御ハザード(control hazard)
    - 分岐命令，ジャンプ命令によって生じるハザード

- まずは，データ・ハザードと制御ハザードが無いものとしてパイプラインプロセッサを構築する．その後，これらに対応するための修正を加える．

戦略4：遅延分岐 (delayed branch), MIPSが採用

- 分岐命令の後続の幾つかの命令を実行した後に，分岐する．1サイクルの遅延を持つ命令実行順は次の通り．
  - 分岐命令を実行
  - 分岐命令の次アドレスの命令を実行
  - 分岐成立では，飛び先アドレスの命令を実行
    （不成立では，分岐命令の次の次のアドレスの命令を実行）
- 分岐命令の後続の幾つかの命令を実行した後に分岐．分岐命令によるストールは生じない．

Untaken分岐命令	IF	ID	EX	MEM	WB				
分岐遅延命令 ($i+1$)		IF	ID	EX	MEM	WB			
命令 $i+2$			IF	ID	EX	MEM	WB		
命令 $i+3$				IF	ID	EX	MEM	WB	
命令 $i+4$					IF	ID	EX	MEM	WB

Taken分岐命令	IF	ID	EX	MEM	WB				
分岐遅延命令 ($i+1$)		IF	ID	EX	MEM	WB			
分岐先			IF	ID	EX	MEM	WB		
分岐先 + 1				IF	ID	EX	MEM	WB	
分岐先 + 2					IF	ID	EX	MEM	WB

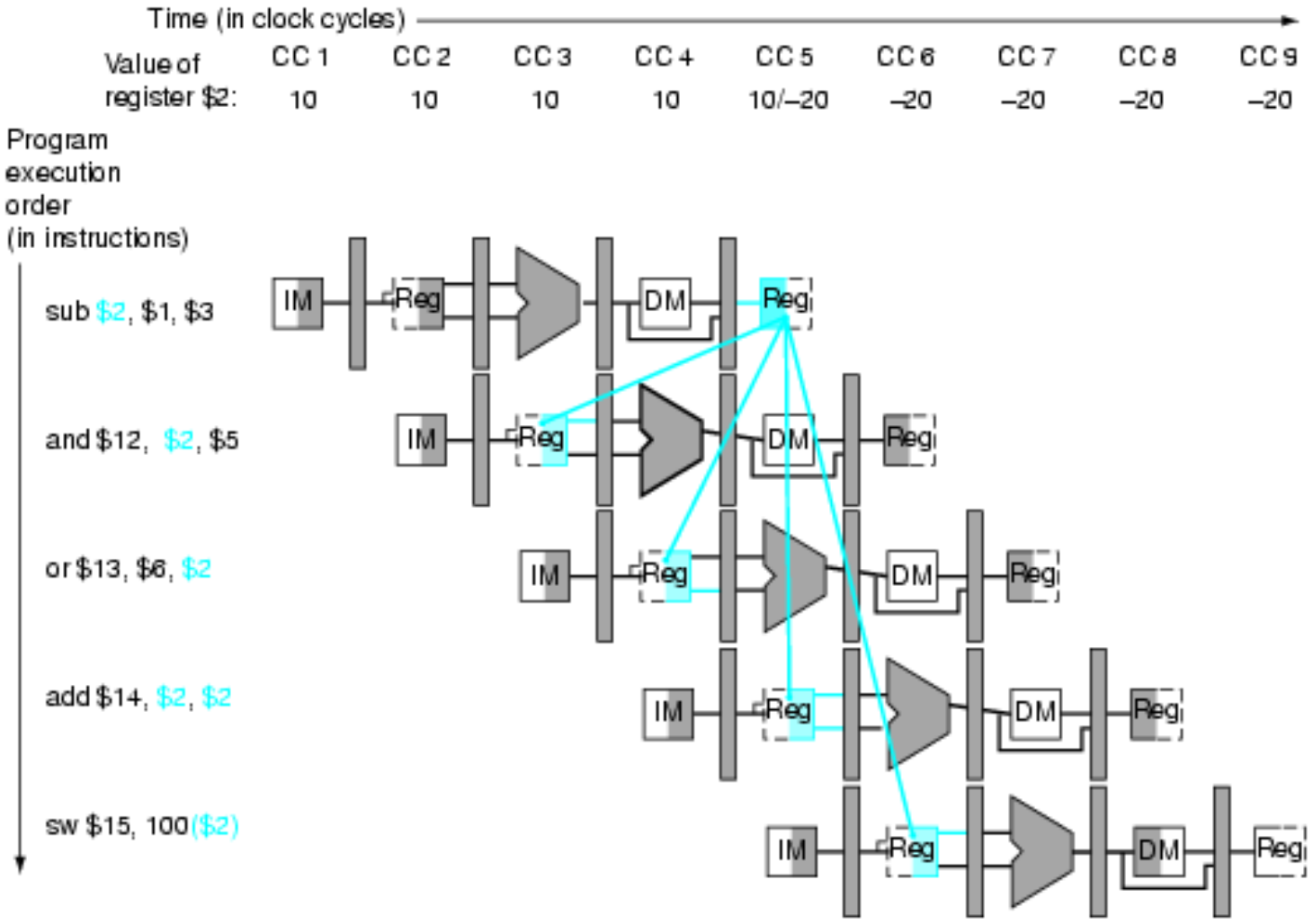
Hazards make pipelining hard

- 命令を適切なサイクルで実行できないような状況が存在する．これをハザード(hazard)と呼ぶ．
  - 構造ハザード (structural hazard)
    - オーバラップ実行する命令の組み合わせをハードウェアがサポートしていない場合．資源不足により生じる．
  - データ・ハザード(data hazard)
    - データの受け渡しの制約によって生じるハザード
  - 制御ハザード(control hazard)
    - 分岐命令，ジャンプ命令によって生じるハザード

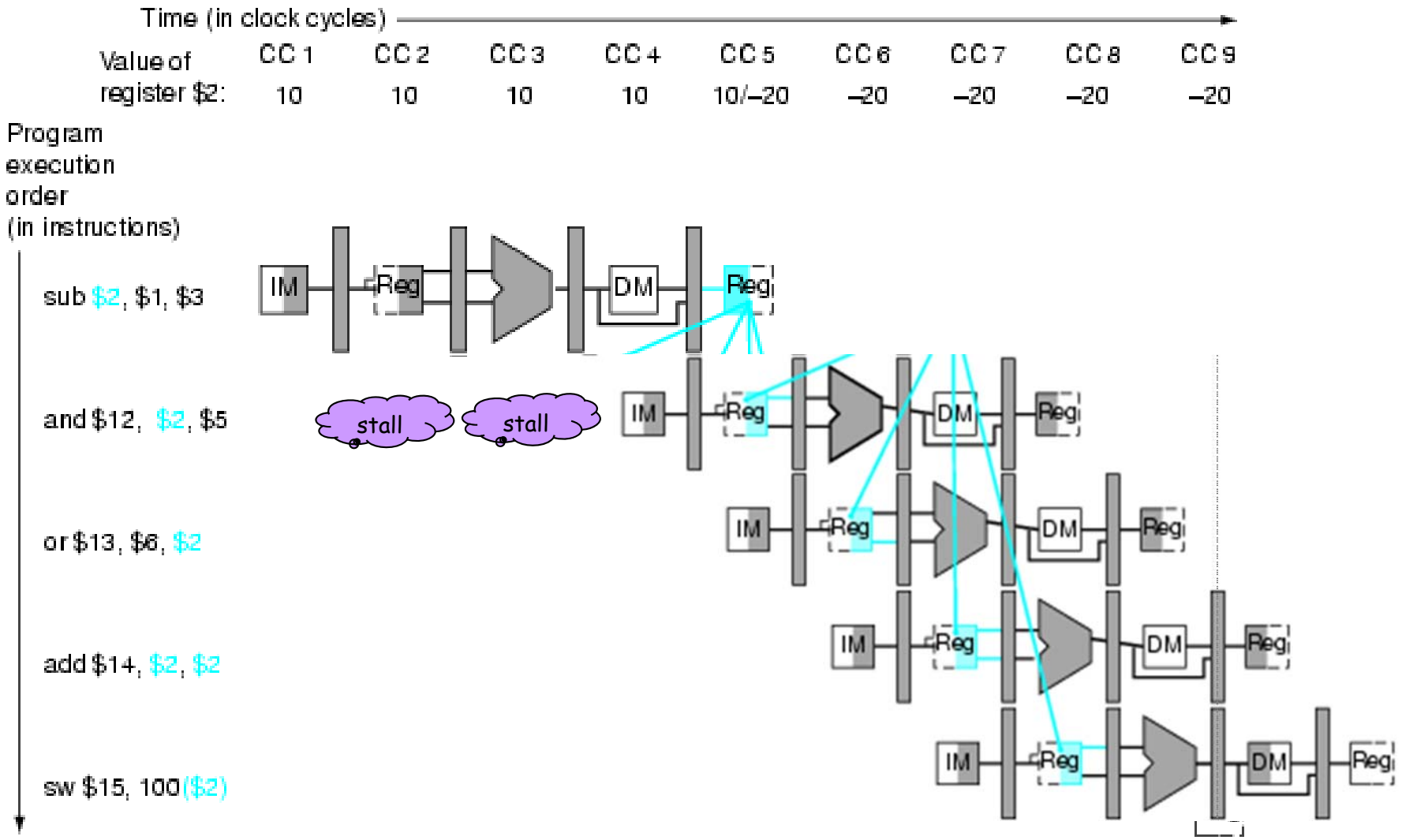
- まずは，データ・ハザードと制御ハザードが無いものとしてパイプラインプロセッサを構築する．その後，これらに対応するための修正を加える．

Data Hazard

- sub命令が生成した$2を後続の命令が利用する場合にデータの受け渡しの制約が生じる.



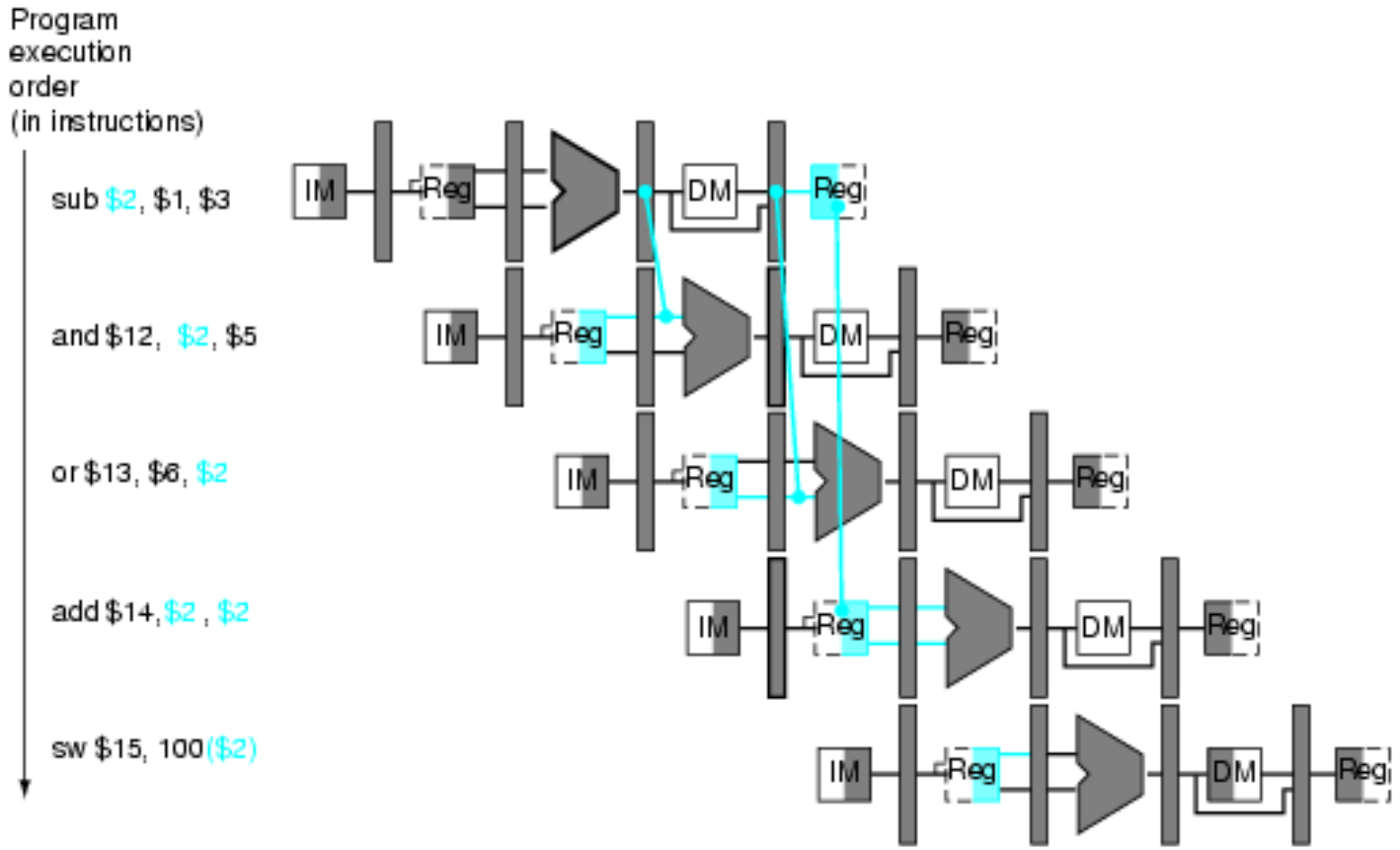
Data Hazard and Stall



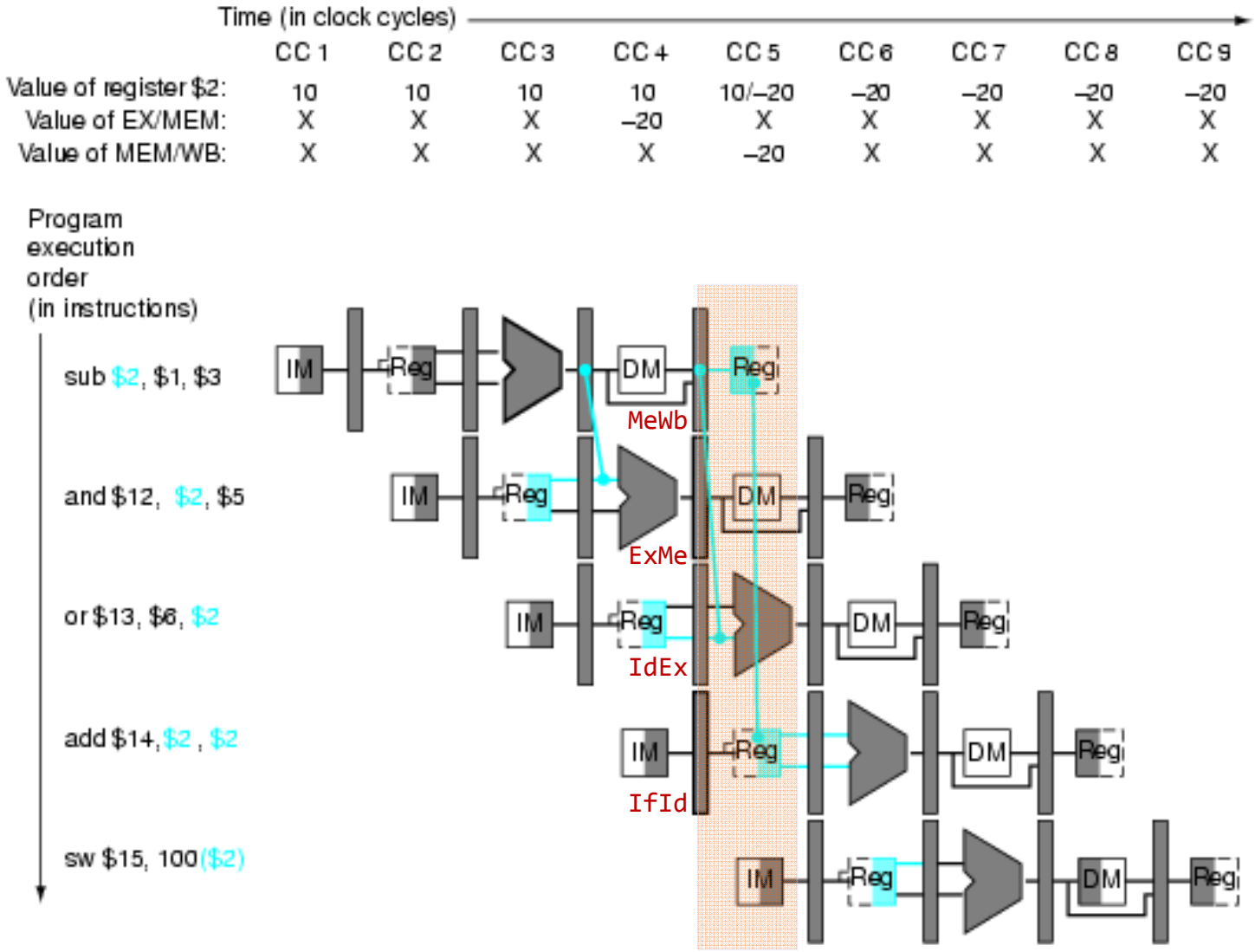
フォワーディングによるデータハザードの回避

Time (in clock cycles)

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X



フォワーディングによるデータハザードの回避



フォワーディングによるデータハザードの回避

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X

Program execution order (in instructions)

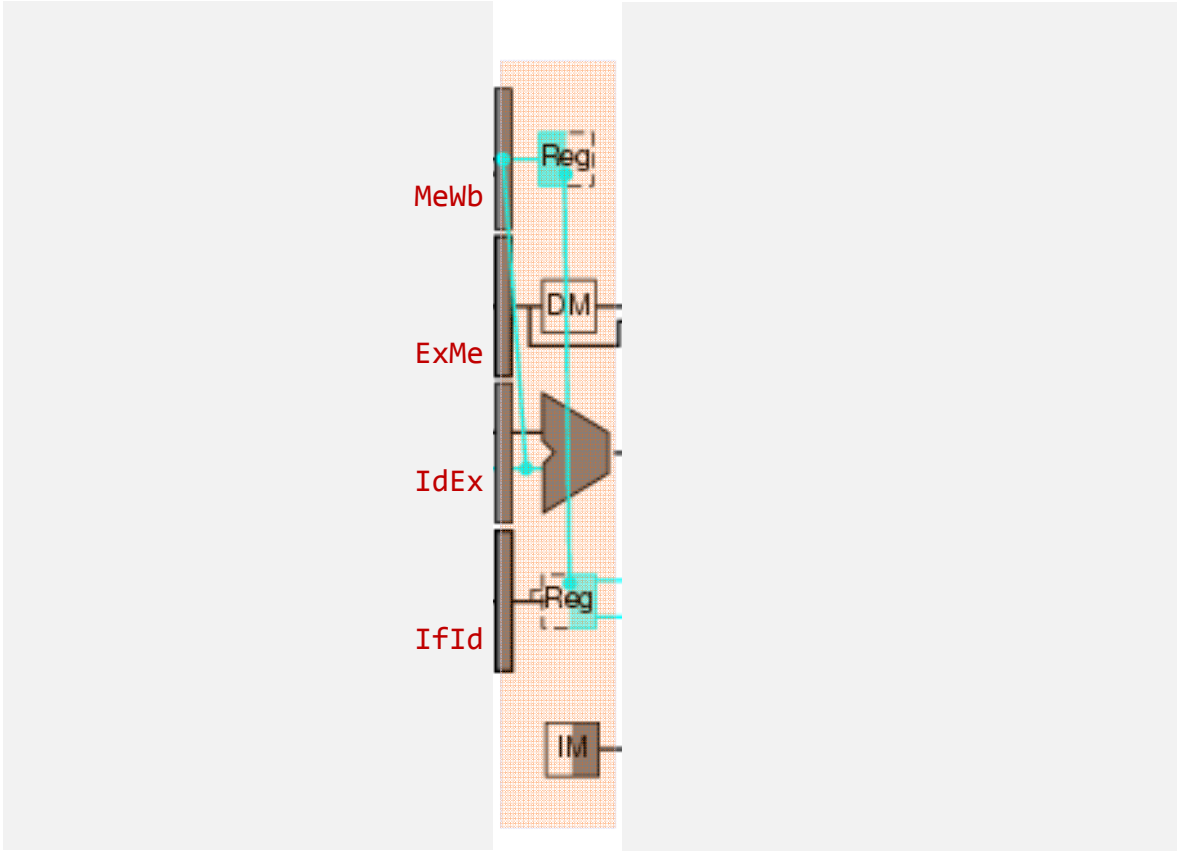
sub $2, $1, $3

and $12, $2, $5

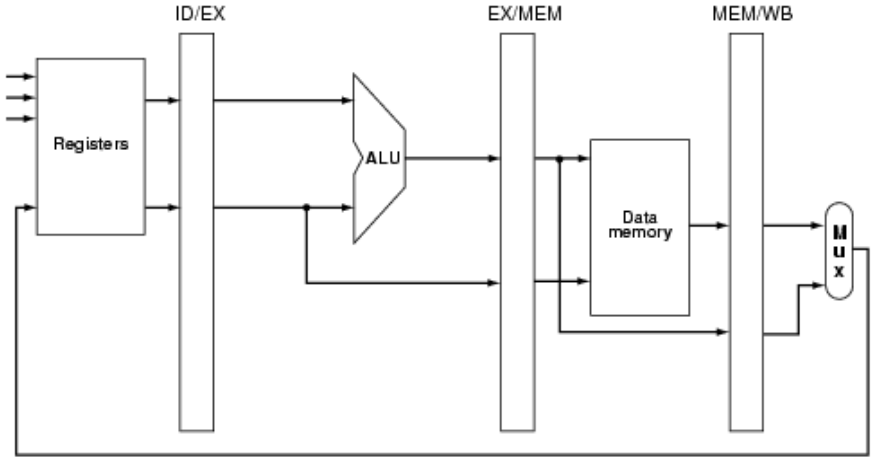
or $13, $6, $2

add $14, $2 , $2

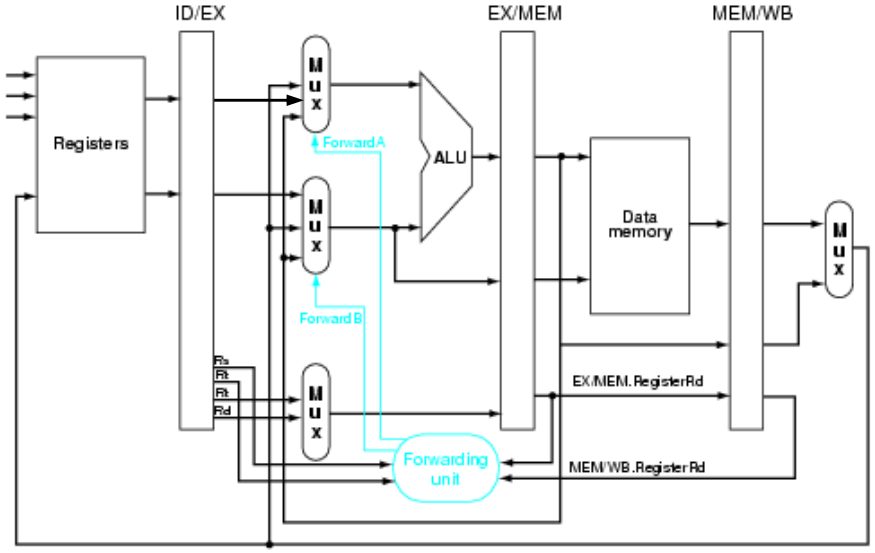
sw $15, 100($2)



フォワーディングのための変更点

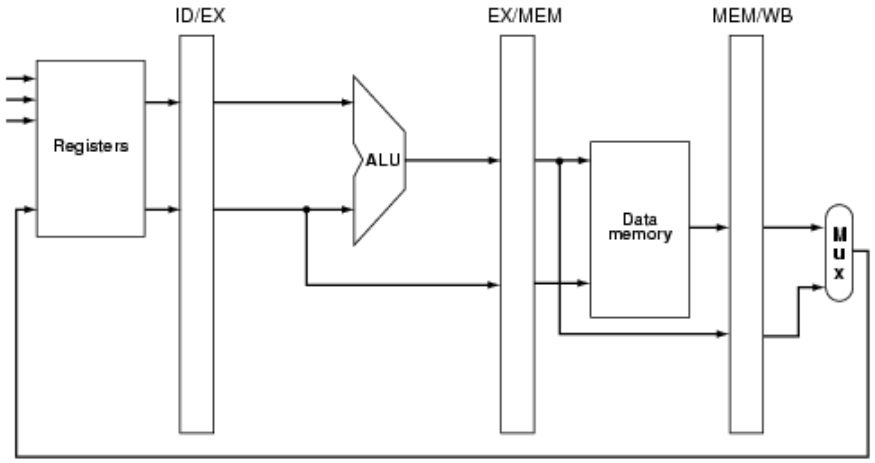


a. No forwarding

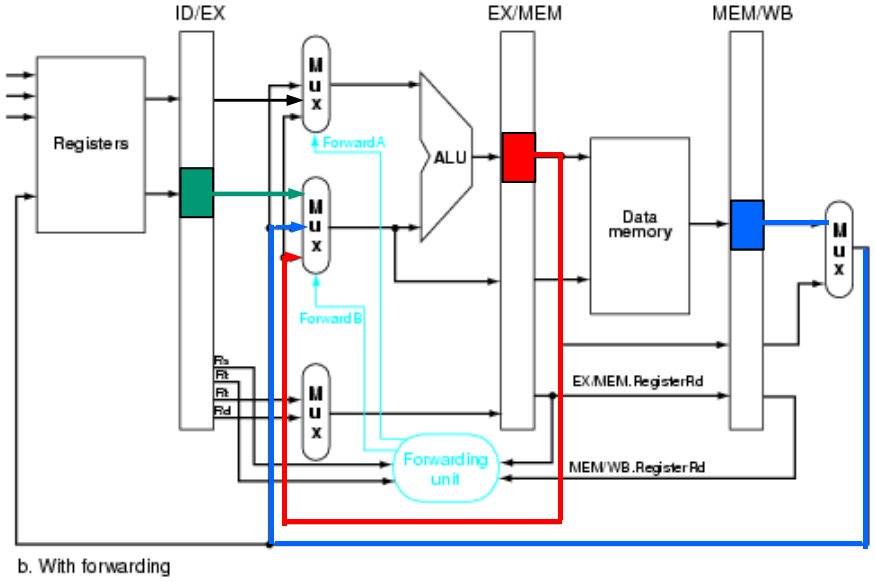


b. With forwarding

フォワーディングのための変更点

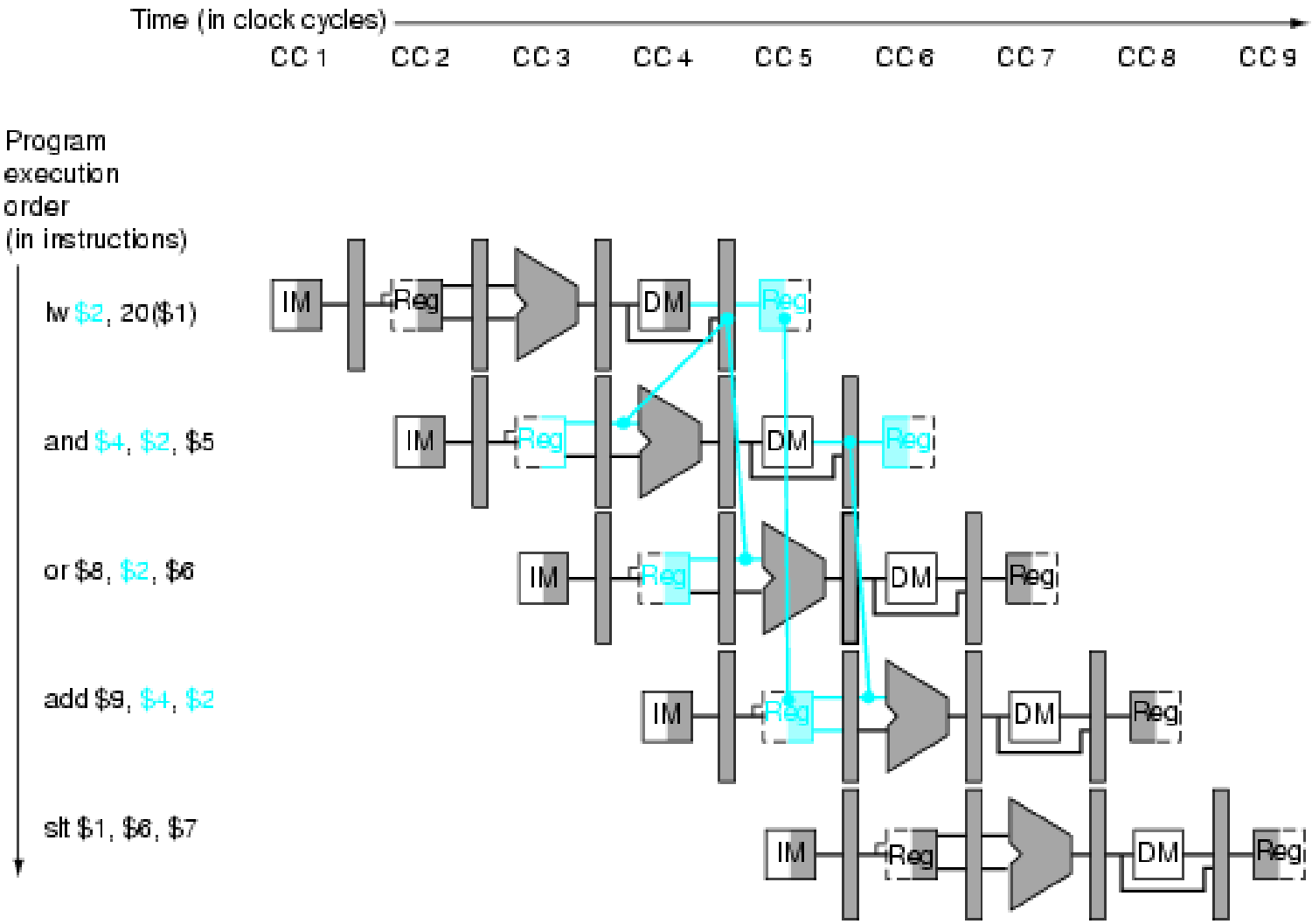


a. No forwarding



b. With forwarding

データハザードによって生じるストール



プロセッサMIPSCORE: 命令フェッチ

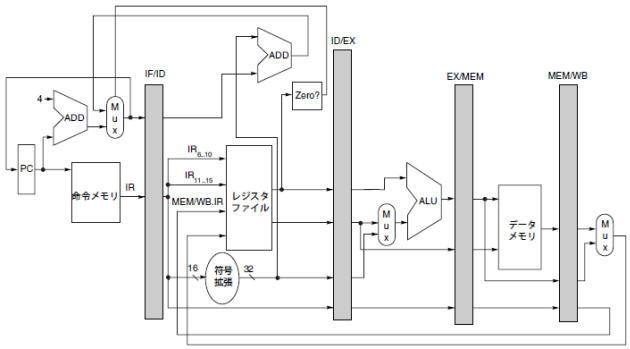
- IdTPC, IdC, IdB はレジスタとして宣言されているが，実際にはワイヤとなる点に注意.
- PSTALL はDRAMアクセスによるプロセッサのストール
- bstall は？

```
41  /*****************************************************************************************************/
42  reg [`ADDR] IdTPC;        // wire, calculated branch Taken address (Taken Program Counter)
43  reg         IdC;          // wire, Calculated branch result, branch Taken or Untaken
44  reg         IdB;          // wire, branch/jump instruction ?
45  wire [31:0] MaRSLT;       // wire, execution result on MA stage
46  wire        bstall;       // stall signal by branch instruction
47  wire        PSTALL = STALL;  // pipeline stall
48
49  /*****************************************************************************************************/
50  /* Stage 1 : IF, instruction fetch                                                                   */
51  /*****************************************************************************************************/
52  wire [`ADDR] IfNPC = pc + 4;
53
54  always @( posedge CLK ) begin ///// update program counter
55      if(!RST_X)                    pc <= 0;
56      else if(!PSTALL && !bstall) pc <= (IdC) ? IdTPC : IfNPC; // If branch taken, uses taken PC
57  end
58
59  always @( posedge CLK ) begin  ///// update pipeline registers
60      if(!RST_X) {IfId_npc, IfId_ir} <= 0;
61      else if(!PSTALL && !bstall) begin
62          IfId_npc <= IfNPC;
63          IfId_ir  <= I_IN;  // if branch taken then NOP else instruction from imem.
64      end
65  end
```

プロセッサMIPSCORE: 命令フェッチ

- IdTPC, IdC, IdB はレジスタとして宣言されているが，実際にはワイヤとなる点に注意.
- PSTALL はDRAMアクセスによるプロセッサのストール
- bstall は？

```
67      /*************************************************************************************************/
68      /* Stage 2 : ID, instruction decode & operand fetch                                              */
69      /*************************************************************************************************/
70      wire [5:0] IdOP   = IfId_ir[31:26]; // opcode field of instruction
71      wire [4:0] IdRS   = IfId_ir[25:21]; // rs field of instruction
72      wire [4:0] IdRT   = IfId_ir[20:16]; // rt field of instruction
73      wire [4:0] IdRD   = IfId_ir[15:11]; // rd field of instruction
74      wire [5:0] IdFCT  = IfId_ir[5:0];   // funct field of instruction
75      wire [31:0] IdRRS, IdRRT;           // register value of rs and rt
76
77      assign bstall = IdB &&
78                  ((IdRS!=0 && (IdRS==IdEx_dst || IdRS==ExMa_dst)) ||
79                   (IdRT!=0 && (IdRT==IdEx_dst || IdRT==ExMa_dst))); // branch stall
```




```
151     always @(posedge CLK) begin ///// update pipeline registers
152         if(!RST_X) {IdEx_npc, IdEx_rrs, IdEx_rrt, IdEx_dst, IdEx_ir, IdEx_opn, IdEx_attr} <= 0;
153         else if(!PSTALL) begin
154             IdEx_npc   <= (bstall) ? 0 : IfId_npc;
155             IdEx_rrs   <= (bstall) ? 0 : IdRRS;    // data from general-purpose register file
156             IdEx_rrt   <= (bstall) ? 0 : IdRRT;    // data from general-purpose register file
157             IdEx_dst   <= (bstall) ? 0 : IdDST;
158             IdEx_ir    <= (bstall) ? 0 : IfId_ir;
159             IdEx_opn   <= (bstall) ? 0 : IdOPN;
160             IdEx_attr  <= (bstall) ? 0 : IdATTR;
161         end
162     end
```

project_24のプロセッサがサポートする命令

- コンテストでは，define.v に定義される次の命令を処理できるプロセッサを実装すること.

```
`define SLL_____   6'd02
`define SRL_____   6'd03
`define SRA_____   6'd04
`define SLLV____   6'd05
`define SRLV____   6'd06
`define SRAV____   6'd07
`define JR______   6'd08
`define JALR____   6'd09
`define ADD_____   6'd14
`define ADDU____   6'd15
`define SUB_____   6'd16
`define SUBU____   6'd17
`define AND_____   6'd18
`define OR______   6'd19
`define XOR_____   6'd20
`define NOR_____   6'd21
`define SLT_____   6'd22
`define SLTU____   6'd23
```

```
`define J_______   6'd24
`define JAL_____   6'd25
`define BEQ_____   6'd26
`define BNE_____   6'd27
`define ADDI____   6'd28
`define ADDIU___   6'd29
`define SLTI____   6'd30
`define SLTIU___   6'd31
`define ANDI____   6'd32
`define ORI_____   6'd33
`define XORI____   6'd34
`define LUI_____   6'd35
`define LW______   6'd38
`define SW______   6'd43
`define BLEZ____   6'd44
`define BGTZ____   6'd45
`define BLTZ____   6'd46
`define BGEZ____   6'd47
```

プロセッサMIPSCORE: 命令デコード

- ビットの切り出し.
- 命令番号 IdOPNの設定.
- 書き込みレジスタ番号 IdDSTの設定.
- この実装では, IdATTR は利用しない.
- レジスタファイルにアクセスして
  オペランドをフェッチする.

```
67  /*****************************************************************************************/
68  /* Stage 2 : ID, instruction decode & operand fetch                                      */
69  /*****************************************************************************************/
70  wire [5:0] IdOP   = IfId_ir[31:26]; // opcode field of instruction
71  wire [4:0] IdRS   = IfId_ir[25:21]; // rs field of instruction
72  wire [4:0] IdRT   = IfId_ir[20:16]; // rt field of instruction
73  wire [4:0] IdRD   = IfId_ir[15:11]; // rd field of instruction
74  wire [5:0] IdFCT  = IfId_ir[5:0];   // funct field of instruction
75  wire [31:0] IdRRS, IdRRT;           // register value of rs and rt
76
77  assign bstall = IdB &&
78              ((IdRS!=0 && (IdRS==IdEx_dst || IdRS==ExMa_dst)) ||
79               (IdRT!=0 && (IdRT==IdEx_dst || IdRT==ExMa_dst))); // branch stall
```

```
81  GPR gpr(.CLK(CLK), .REGNUM0(IdRS), .REGNUM1(IdRT), .DOUT0(IdRRS), .DOUT1(IdRRT), // register file
82          .REGNUM2(ExMa_dst), .DIN0(MaRSLT), .WE0(!PSTALL));
83
84  reg [6:0]   IdOPN;   // instruction operation number (unique operation ID)
85  reg [4:0]   IdDST;   // destination register
86  reg [`ATTR] IdATTR;  // instruction attribute
87  always @(IfId_ir) begin
88      IdOPN   = `ERROR___;
89      IdDST  = 0;
90      IdATTR = 0;
91      case (IdOP) // OP
92          6'h00: case (IdFCT) // FUNCT
93              6'h00: begin IdOPN= (IdRD) ? `SLL_____ : `NOP_____; IdDST=IdRD; end
94              6'h02: begin IdOPN=`SRL_____; IdDST=IdRD;                    end
95              6'h03: begin IdOPN=`SRA_____; IdDST=IdRD;                    end
96              6'h04: begin IdOPN=`SLLV____; IdDST=IdRD;                    end
97              6'h06: begin IdOPN=`SRLV____; IdDST=IdRD;                    end
98              6'h07: begin IdOPN=`SRAV____; IdDST=IdRD;                    end
99              6'h08: begin IdOPN=`JR______; IdDST=0;                       end
100             6'h09: begin IdOPN=`JALR____; IdDST=31;                      end
101             6'h20: begin IdOPN=`ADD_____; IdDST=IdRD;                    end
102             6'h21: begin IdOPN=`ADDU____; IdDST=IdRD;                    end
103             6'h22: begin IdOPN=`SUB_____; IdDST=IdRD;                    end
104             6'h23: begin IdOPN=`SUBU____; IdDST=IdRD;                    end
105             6'h24: begin IdOPN=`AND_____; IdDST=IdRD;                    end
106             6'h25: begin IdOPN=`OR______; IdDST=IdRD;                    end
107             6'h26: begin IdOPN=`XOR_____; IdDST=IdRD;                    end
108             6'h27: begin IdOPN=`NOR_____; IdDST=IdRD;                    end
109             6'h2a: begin IdOPN=`SLT_____; IdDST=IdRD;                    end
110             6'h2b: begin IdOPN=`SLTU____; IdDST=IdRD;                    end
111         endcase
112         6'h01: case (IdRT)
113             5'h00: begin IdOPN=`BLTZ____; IdDST=0;                       end
114             5'h01: begin IdOPN=`BGEZ____; IdDST=0;                       end
115         endcase
116         6'h02:     begin IdOPN=`J_______; IdDST=0;                       end
117         6'h03:     begin IdOPN=`JAL_____; IdDST=31;                      end
118         6'h04:     begin IdOPN=`BEQ_____; IdDST=0;                       end
119         6'h05:     begin IdOPN=`BNE_____; IdDST=0;                       end
120         6'h06:     begin IdOPN=`BLEZ____; IdDST=0;                       end
121         6'h07:     begin IdOPN=`BGTZ____; IdDST=0;                       end
122         6'h08:     begin IdOPN=`ADDI____; IdDST=IdRT;                    end
123         6'h09:     begin IdOPN=`ADDIU___; IdDST=IdRT;                    end
124         6'h0a:     begin IdOPN=`SLTI____; IdDST=IdRT;                    end
125         6'h0b:     begin IdOPN=`SLTIU___; IdDST=IdRT;                    end
126         6'h0c:     begin IdOPN=`ANDI____; IdDST=IdRT;                    end
127         6'h0d:     begin IdOPN=`ORI_____; IdDST=IdRT;                    end
128         6'h0e:     begin IdOPN=`XORI____; IdDST=IdRT;                    end
129         6'h0f:     begin IdOPN=`LUI_____; IdDST=IdRT;                    end
130         6'h23:     begin IdOPN=`LW______; IdDST=IdRT; IdATTR=`LD_4B;     end
131         6'h2b:     begin IdOPN=`SW______; IdDST=0;    IdATTR=`ST_4B;     end
132     endcase
133 end
```

プロセッサMIPSCORE: 命令デコード

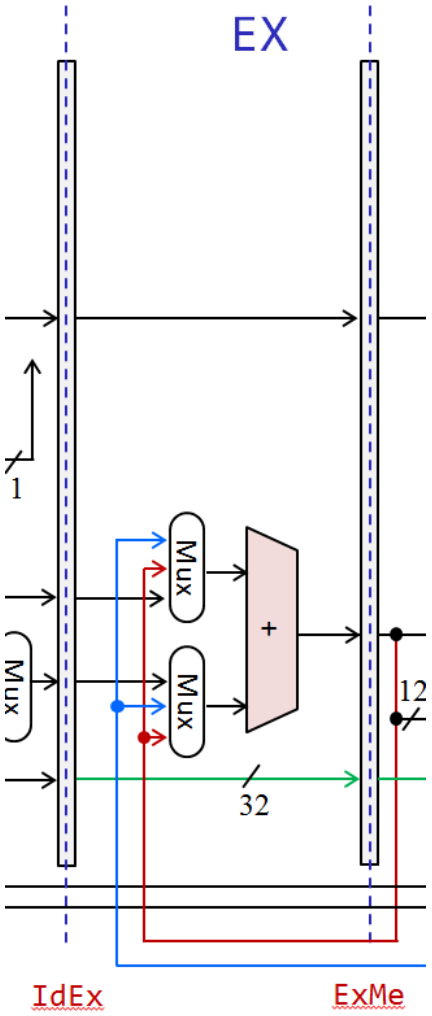
- 分岐命令の実行.

```
41      /*****************************************************************************
42      reg [`ADDR] IdTPC;        // wire, calculated branch Taken address (Taken Program Counter)
43      reg         IdC;          // wire, Calculated branch result, branch Taken or Untaken
44      reg         IdB;          // wire, branch/jump instruction ?
45      wire [31:0] MaRSLT;       // wire, execution result on MA stage
46      wire        bstall;       // stall signal by branch instruction
47      wire        PSTALL = STALL;  // pipeline stall
```

```
135     wire [`ADDR] IdBPC = IfId_npc + ({{16{IfId_ir[15]}}, IfId_ir[15:0]} << 2);
136     always @(*) begin  ////// branch & jump resolution unit
137         {IdTPC, IdC, IdB} = 0;
138         case (IdOP)
139             6'h04: begin IdB=1; IdTPC = IdBPC; IdC = (IdRRS == IdRRT);         end        // BEQ
140             6'h05: begin IdB=1; IdTPC = IdBPC; IdC = (IdRRS != IdRRT);         end        // BNE
141             6'h06: begin IdB=1; IdTPC = IdBPC; IdC = ( IdRRS[31]||(IdRRS==0)); end        // BLEZ
142             6'h07: begin IdB=1; IdTPC = IdBPC; IdC = (~IdRRS[31]&&(IdRRS!=0)); end        // BGTZ
143             6'h01: begin IdB=1; IdTPC = IdBPC; IdC = (IdRT) ? ~IdRRS[31] : IdRRS[31]; end // BGEZ,BLTZ
144             6'h02: begin IdB=1; IdTPC = {IfId_npc[31:28], IfId_ir[25:0], 2'b0}; IdC=1;end // J
145             6'h03: begin IdB=1; IdTPC = {IfId_npc[31:28], IfId_ir[25:0], 2'b0}; IdC=1;end // JAL
146             6'h00: if      (IdFCT==6'h08) begin IdB=1; IdTPC = IdRRS; IdC = 1; end        // JR
147                    else if (IdFCT==6'h09) begin IdB=1; IdTPC = IdRRS; IdC = 1; end        // JALR
148         endcase
149     end
150
151     always @(posedge CLK) begin ///// update pipeline registers
152         if(!RST_X) {IdEx_npc, IdEx_rrs, IdEx_rrt, IdEx_dst, IdEx_ir, IdEx_opn, IdEx_attr} <= 0;
153         else if(!PSTALL) begin
154             IdEx_npc   <= (bstall) ? 0 : IfId_npc;
155             IdEx_rrs   <= (bstall) ? 0 : IdRRS;    // data from general-purpose register file
156             IdEx_rrt   <= (bstall) ? 0 : IdRRT;    // data from general-purpose register file
157             IdEx_dst   <= (bstall) ? 0 : IdDST;
158             IdEx_ir    <= (bstall) ? 0 : IfId_ir;
159             IdEx_opn   <= (bstall) ? 0 : IdOPN;
160             IdEx_attr  <= (bstall) ? 0 : IdATTR;
161         end
162     end
```

プロセッサMIPSCORE: 実行

- データフォワーディングと実行




```
164     /*************************************************************************************/
165     /* Stage 3 : EX, execute & address generation for LD/ST                              */
166     /*************************************************************************************/
167     wire        [31:0] RRS_U, RRT_U;                    // output of data forwarding unit
168     wire signed [31:0] RRS_S = RRS_U;                   // signed wire
169     wire signed [31:0] RRT_S = RRT_U;                   // signed wire
170     wire         [4:0] SHAMT  = IdEx_ir[10:6];          // Shift amount
171     wire        [15:0] IMM    = IdEx_ir[15:0];          // Immediate
172     wire        [31:0] SET32I = {{16{IMM[15]}}, IMM};   // Sign Extended Imm.
173     wire       [`ADDR] SETADI = SET32I[`ADDR] << 2;     // shifted immediate of address bit width
174     wire       [`ADDR] JADDR  = IdEx_ir[`ADDR] << 2;    // Juma address
175     wire       [`ADDR] ExA = RRS_U[`ADDR] + SET32I[`ADDR]; // mem address of LD/ST
176     reg         [31:0] ExRSLT;                          // execution result
177     reg          [3:0] ExWE;                            // memory write enable vector
178
179     FORWARDING frs(.SRC(IdEx_ir[25:21]), .DIN0(IdEx_rrs), .DOUT(RRS_U),
180                    .DST1(ExMa_dst), .DIN1(ExMa_rslt), .DST2(MaWb_dst), .DIN2(MaWb_rslt));
181     FORWARDING frt(.SRC(IdEx_ir[20:16]), .DIN0(IdEx_rrt), .DOUT(RRT_U),
182                    .DST1(ExMa_dst), .DIN1(ExMa_rslt), .DST2(MaWb_dst), .DIN2(MaWb_rslt));
183
184     always @(*) begin
185         {ExRSLT, ExWE} = 0;
186         case ( IdEx_opn )
187           `ADD_____ : begin ExRSLT = RRS_U + RRT_U;                                    end
188           `ADDI____ : begin ExRSLT = RRS_U + SET32I;                                   end
189           `ADDIU___ : begin ExRSLT = RRS_U + SET32I;                                   end
190           `ADDU____ : begin ExRSLT = RRS_U + RRT_U;                                    end
191           `SUB_____ : begin ExRSLT = RRS_U - RRT_U;                                    end
192           `SUBU____ : begin ExRSLT = RRS_U - RRT_U;                                    end
193           `AND_____ : begin ExRSLT = RRS_U & RRT_U;                                    end
194           `ANDI____ : begin ExRSLT = RRS_U & {16'h0, IMM};                             end
195           `NOR_____ : begin ExRSLT = ~(RRS_U | RRT_U);                                 end
196           `OR______ : begin ExRSLT = RRS_U | RRT_U;                                    end
197           `ORI_____ : begin ExRSLT = RRS_U | {16'h0, IMM};                             end
198           `XOR_____ : begin ExRSLT = RRS_U ^ RRT_U;                                    end
199           `XORI____ : begin ExRSLT = RRS_U ^ {16'h0, IMM};                             end
200           `SLL_____ : begin ExRSLT = RRT_U <<  SHAMT;                                  end
201           `SRL_____ : begin ExRSLT = RRT_U >>  SHAMT;                                  end
202           `SRA_____ : begin ExRSLT = RRT_S >>> SHAMT;                                  end
203           `SLLV____ : begin ExRSLT = RRT_U <<  RRS_U[4:0];                             end
204           `SRLV____ : begin ExRSLT = RRT_U >>  RRS_U[4:0];                             end
205           `SRAV____ : begin ExRSLT = RRT_S >>> RRS_U[4:0];                             end
206           `SLT_____ : begin ExRSLT = (RRS_U[31] ^ RRT_U[31]) ? RRS_U[31] : (RRS_U < RRT_U);  end
207           `SLTI____ : begin ExRSLT = (RRS_U[31] ^   IMM[15]) ? RRS_U[31] : (RRS_U < SET32I); end
208           `SLTIU___ : begin ExRSLT = (RRS_U < SET32I);                                 end
209           `SLTU____ : begin ExRSLT = (RRS_U < RRT_U);                                  end
210           `JAL_____ : begin ExRSLT = IdEx_npc + 4;                                     end
211           `JALR____ : begin ExRSLT = IdEx_npc + 4;                                     end
212           `LUI_____ : begin ExRSLT = {IMM, 16'h0};                                     end
213           `LW______ : begin ExRSLT = {ExA[31:2], 2'b00};                               end
214           `SW______ : begin ExRSLT = {ExA[31:2], 2'b00}; ExWE = 4'b1111;               end
215         endcase
216     end
```

プロセッサMIPSCORE: 実行

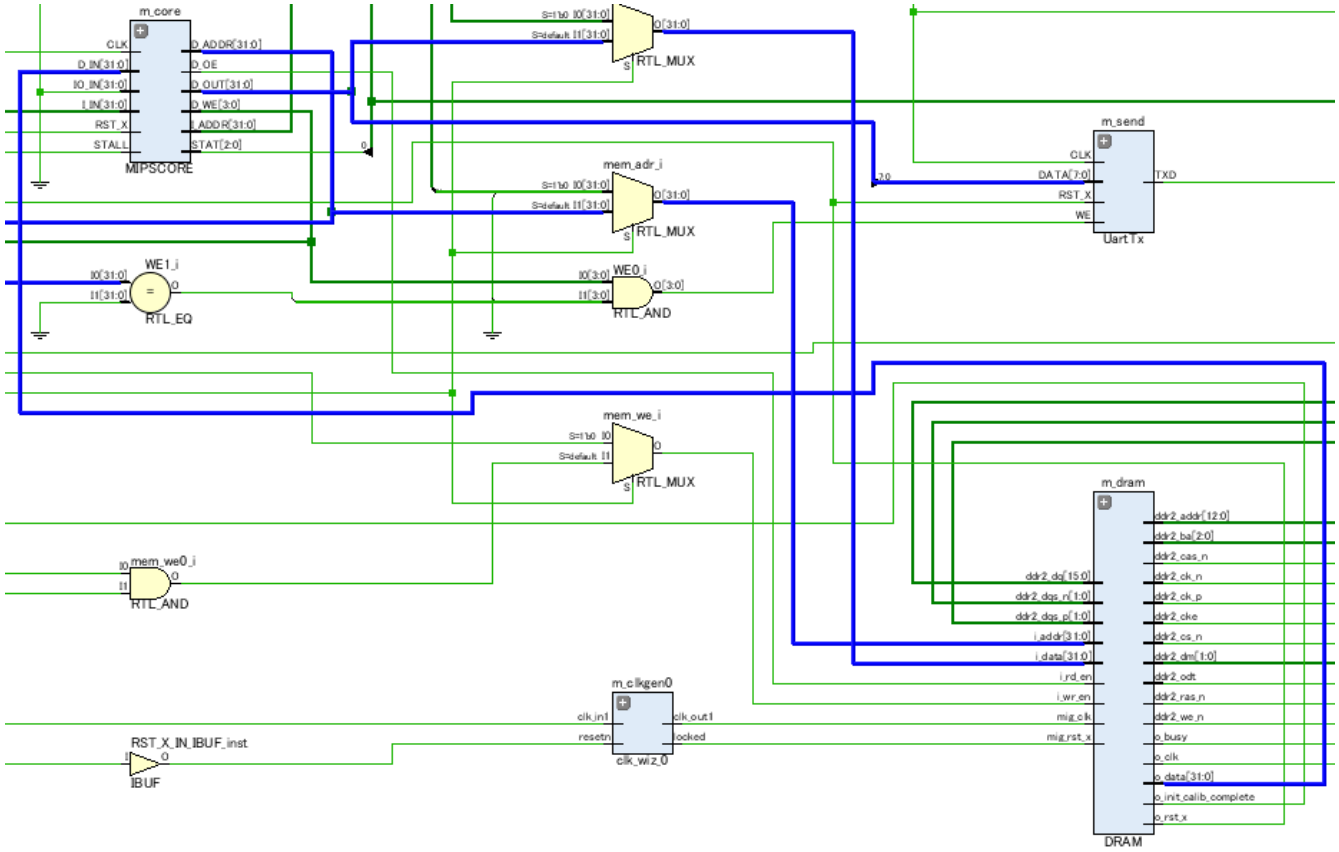
- データフォワーディングと実行

```
282 /***** data forwarding unit                                                                    *****/
283 /*************************************************************************************************/
284 module FORWARDING(SRC, DIN0, DST1, DIN1, DST2, DIN2, DOUT);
285     input  wire  [4:0] SRC, DST1, DST2;   // register number
286     input  wire [31:0] DIN0, DIN1, DIN2;  // 32bit value
287     output wire [31:0] DOUT;              // 32bit data output
288
289     assign DOUT = (SRC!=0 && DST1==SRC) ? DIN1 : (SRC!=0 && DST2==SRC) ? DIN2 : DIN0;
290 endmodule
291 /*************************************************************************************************/
```

```
218     always @( posedge CLK ) begin ///// update pipeline registers
219         if(!RST_X) {ExMa_rslt, ExMa_dst, ExMa_mwe, ExMa_oe, ExMa_lds, ExMa_std} <= 0;
220         else if(!PSTALL) begin
221             ExMa_rslt <= ExRSLT;
222             ExMa_dst  <= IdEx_dst;
223             ExMa_std  <= RRT_U;
224             ExMa_mwe  <= ExWE;
225             ExMa_oe   <= (IdEx_attr & `LDST_ANY) ? 1 : 0;
226             ExMa_lds  <= (IdEx_opn==`LW______) ?  1 : 0;
227         end
228     end
```

Data Memory (DRAM, m_dram)

- プロセッサMIPSCOREはストアすべきデータD_OUTを出力し，それがメモリコントローラDRAMの入力i_dataとなる．また，プロセッサはストアすべきデータのアドレスD_ADDRを出力し，それがメモリコントローラの入力i_addrとなる．プロセッサの出力D_WEにより，DRAMへのデータのストアを指示する．
- プロセッサはロードするアドレスD_ADDRを出力し，それがメモリコントローラの入力i_addrとなる．プロセッサの出力D_OEにより，DRAMへのデータのリードを指示する．DRAMから得られたデータがo_dataに出力され，それがプロセッサの入力D_INとなる．



プロセッサMIPSCORE: メモリアクセス

```
230     /*****************************************************************************************************/
231     /* Stage 4 : MA, memory access for LD/ST                                                             */
232     /*****************************************************************************************************/
233     assign D_ADDR = (ExMa_oe)            ? ExMa_rslt : 0;
234     assign D_OUT  = (ExMa_oe)            ? ExMa_std  : 0;
235     assign D_WE   = (ExMa_oe && !PSTALL) ? ExMa_mwe  : 0;
236     assign D_OE   =  ExMa_oe;
237
238     // wire [31:0] MaLD_ = (ExMa_rslt[23]) ? IO_IN : D_IN; // use I/O input if the high address
239     wire [31:0] MaLD_ = D_IN; // This edition does not use I/O.
240     wire [31:0] MaLDD = MaLD_ >> (8*ExMa_rslt[1:0]);    // loaded data, align data here
241
242     assign MaRSLT = (ExMa_lds) ? MaLDD : ExMa_rslt;
243
244     always @( posedge CLK ) begin ///// update pipeline registers
245         if(!RST_X) {MaWb_rslt, MaWb_dst} <= 0;
246         else if(!PSTALL) begin
247             MaWb_rslt <= MaRSLT;
248             MaWb_dst  <= ExMa_dst;
249         end
250     end
```

リファレンスデザインに含まれるプロセッサ

- Operating frequency: 50MHz
- リファレンスデザインには，5段パイプライン処理の典型的なMIPSプロセッサ（のサブセット）が採用されている．下の図からレジスタ名などが変わっているので注意．