Department of Computer Science
Course number: CSC.T341

コンピュータ論理設計
Computer Logic Design

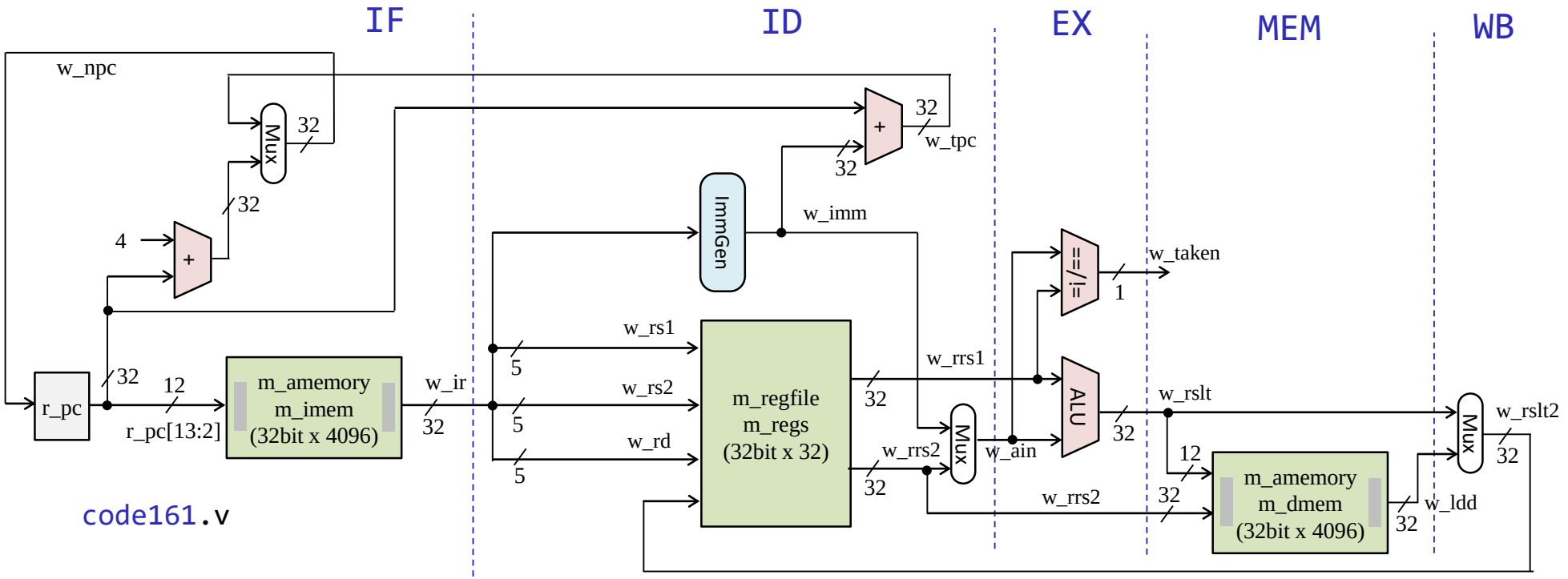
10. マルチサイクルプロセッサの設計と実装
Design and Implementation of a Multi-cycle Processor

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
講義：月曜日 10:45-12:25, 木曜日 10:45-12:25

Ver. 2023-05-15a

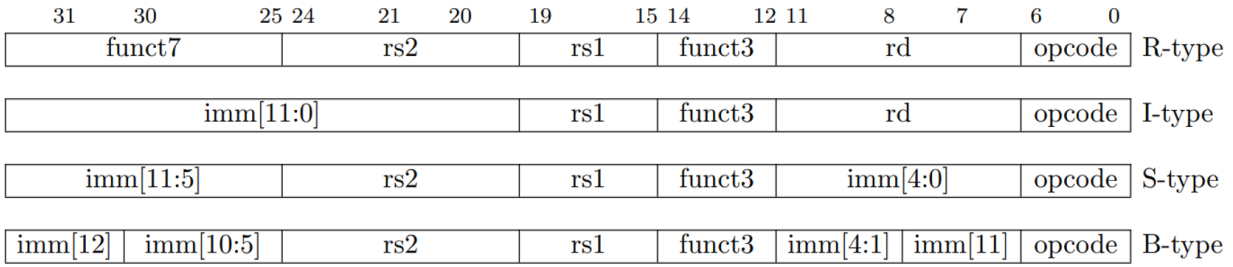
m_proc07 ベースラインのプロセッサ（シングルサイクル）

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), メモリアクセス(MEM), ライトバック(WB) の処理をおこなうadd, addi, sll, srl, lw, sw, beq, bne命令に対応したプロセッサ



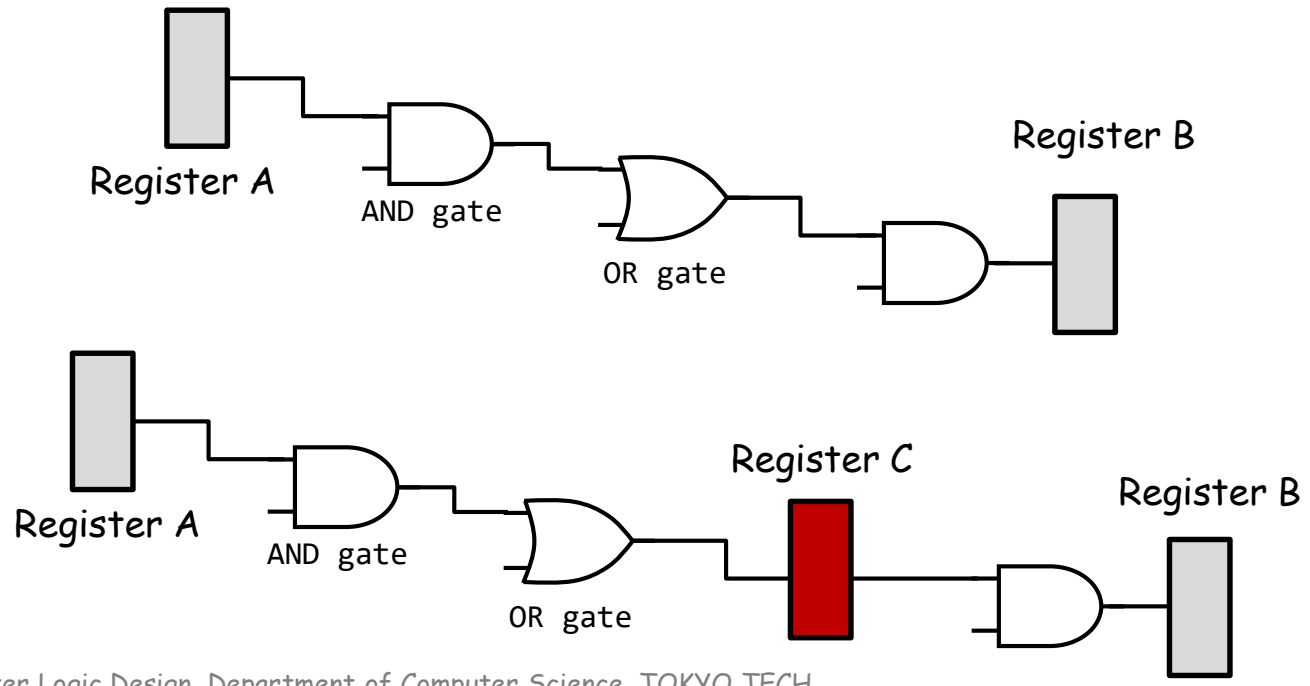
code161.v

```
f    = 60MHz
IPC  = 1.000
Perf = 60.00
```



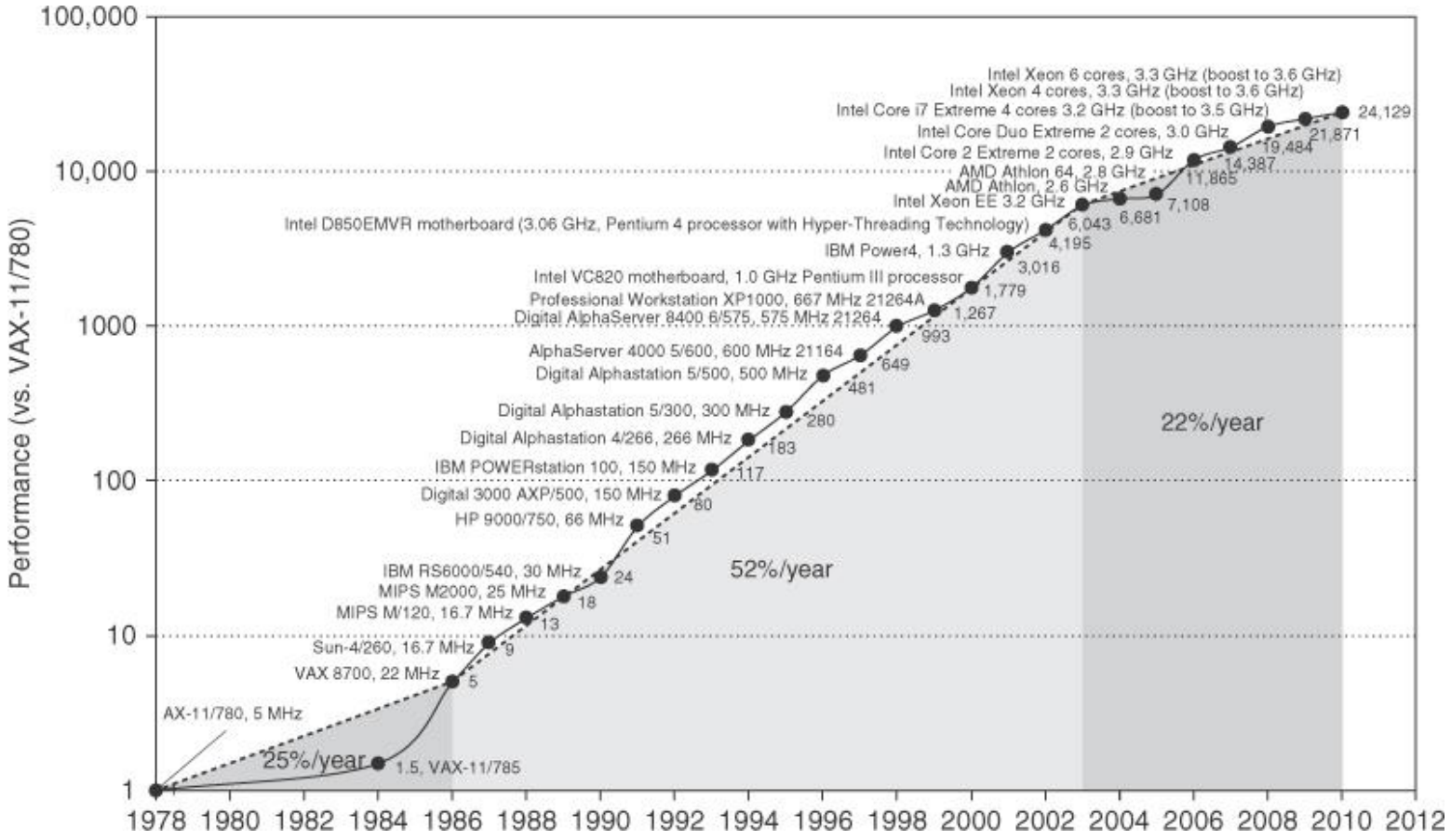
Clock rate is mainly determined by

- Switching speed of gates (transistors)
- **The number of levels of gates**
  - The maximum number of gates cascaded in series in any combinational logics.
  - In this example, the number of levels of gates is 3.
- Wiring delay and fanout



Growth in processor performance

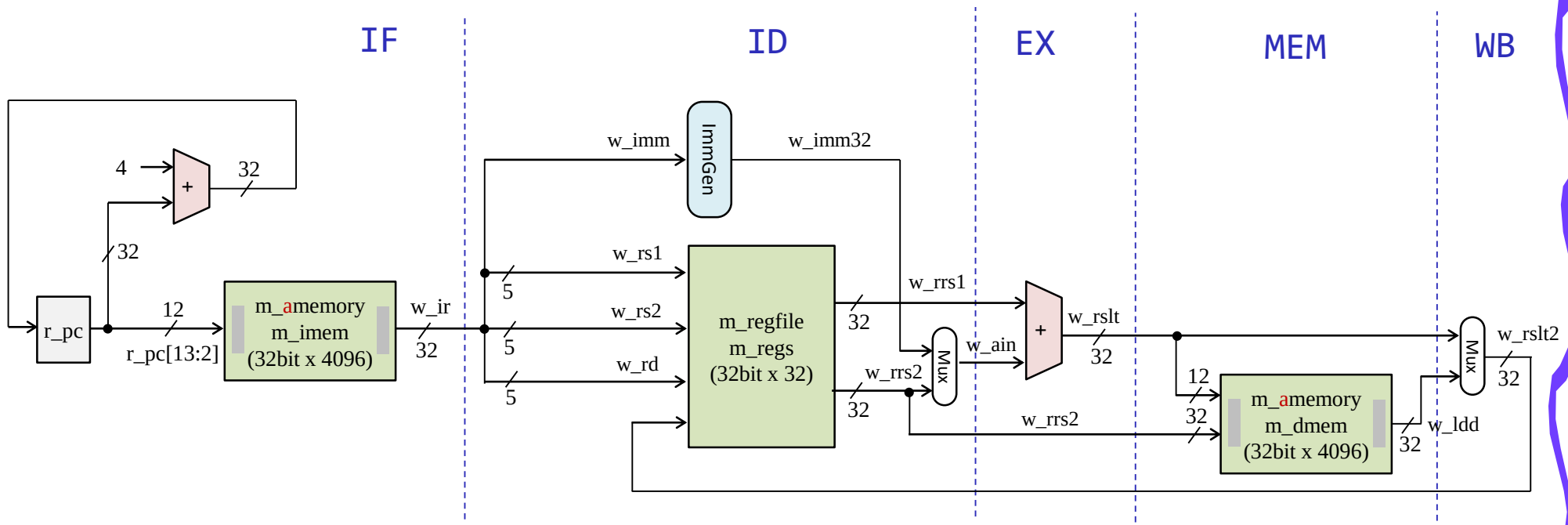
- Performance = f x IPC
  - f: frequency (clock rate)
  - IPC: retired machine Instructions Per Cycle

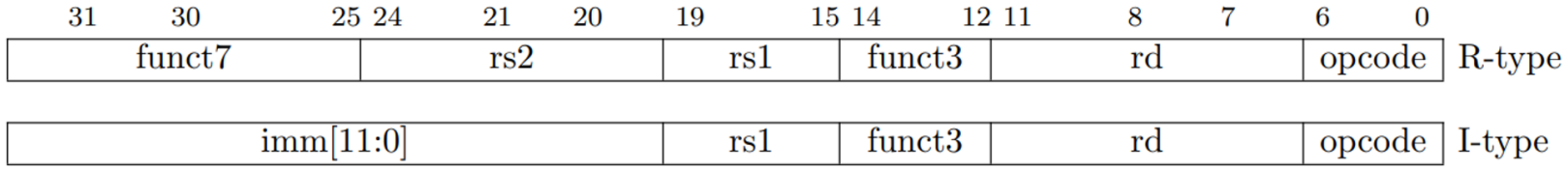


From CAQA 5th edition

m_proc05 add, addi, lw, sw を処理するプロセッサ

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), メモリアクセス(MEM), ライトバック(WB) の処理をおこなうadd, addi, lw, sw命令に対応したプロセッサ



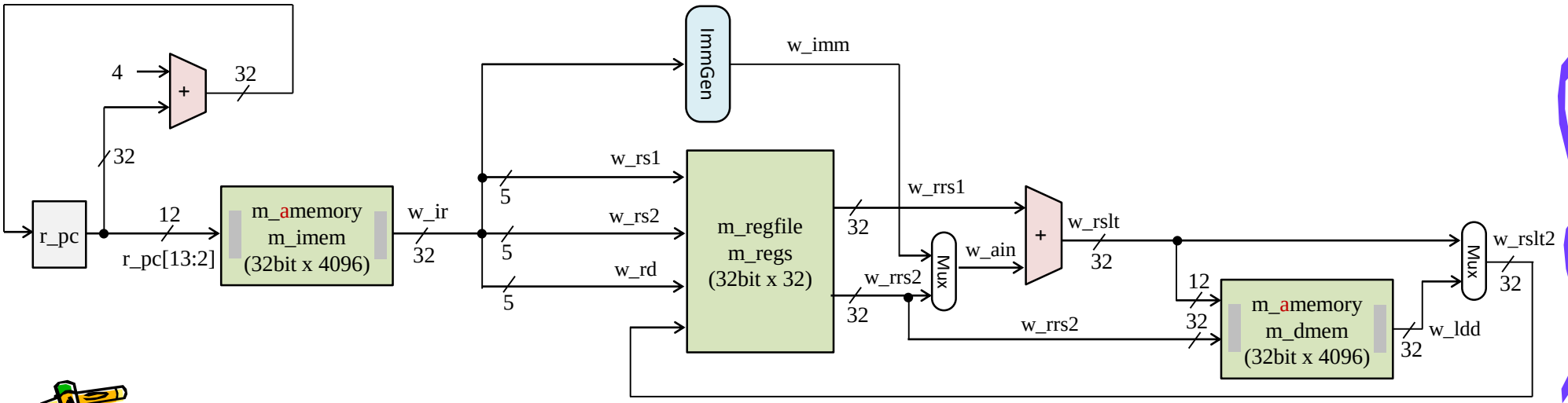


m_proc05 add, addi, lw, sw を処理するシングルサイクル版

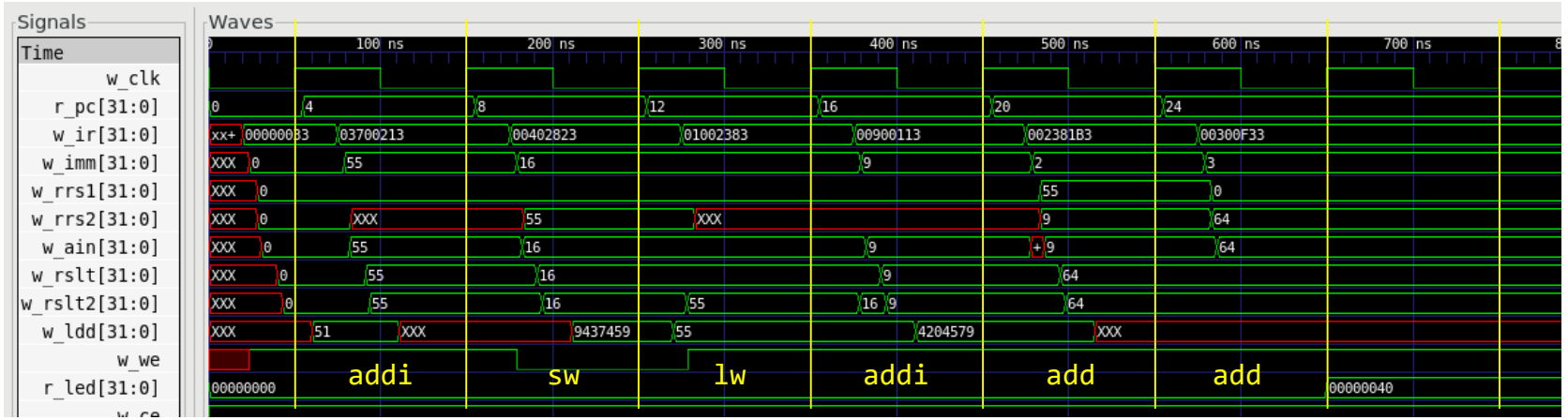
```
module m_proc05 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  wire [31:0] w_ir;
  wire [4:0]  w_op5 = w_ir[6:2];
  wire [4:0]  w_rs1 = w_ir[19:15];
  wire [4:0]  w_rs2 = w_ir[24:20];
  wire [4:0]  w_rd  = w_ir[11:7];
```

code171.v

```
  wire #4 w_we = w_ce & (w_op5==5'b01100 || w_op5==5'b00100 || w_op5==5'b00000);
  wire [31:0] w_imm, w_rrs1, w_rrs2, w_ain, w_rslt, w_ldd, w_rslt2;

  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);
  m_immgen m_immgen0 (w_ir, w_imm);
  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_we, w_rslt2, w_rrs1, w_rrs2);
  assign #3 w_ain = (w_op5==5'b01100) ? w_rrs2 : w_imm;
  assign #9 w_rslt = w_rrs1 + w_ain;
  m_amemory m_dmem (w_clk, w_rslt[13:2], (w_op5==5'b01000), w_rrs2, w_ldd);
  assign #3 w_rslt2 = (w_op5==5'b00000) ? w_ldd : w_rslt;
  always @(posedge w_clk) #5 if(w_ce & r_pc!=24) r_pc <= r_pc + 4;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_we & w_rd==30) r_led <= w_rslt2;
  assign w_led = r_led;
endmodule
```

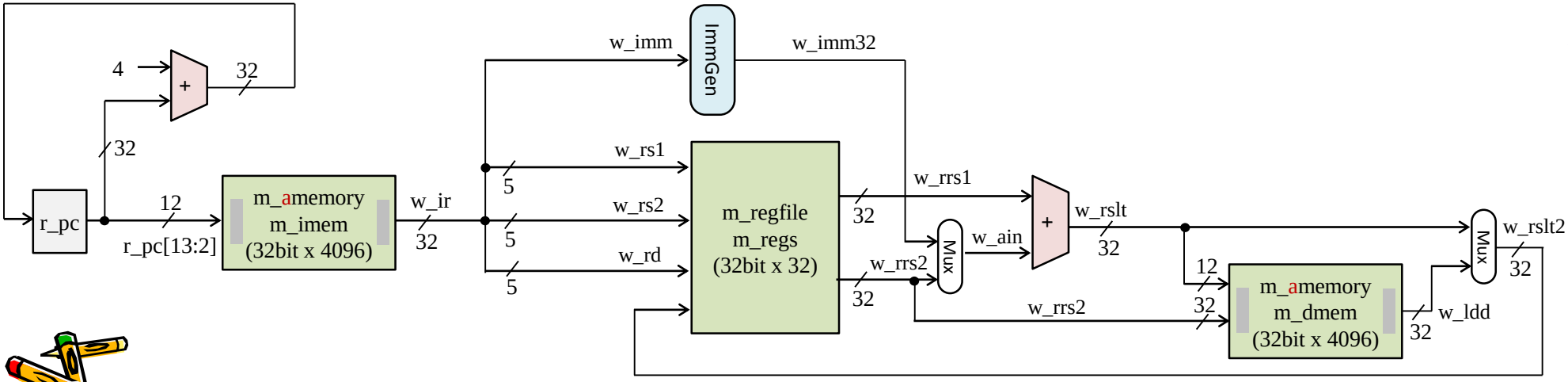


m_proc05 add, addi, lw, sw を処理するシングルサイクル版



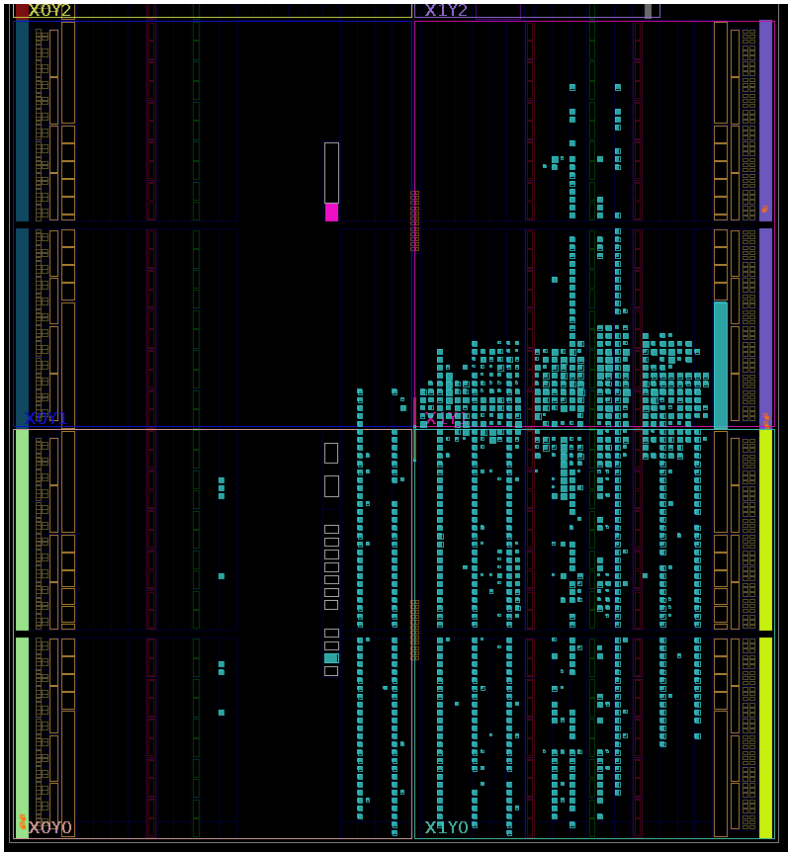

```
cm_ram[0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011}; //    add  x0, x0, x0  // NOP
cm_ram[1]={12'd55,     5'd0, 3'b000, 5'd4, 7'b0010011}; //    addi x4, x0, 55  // x4 = 55
cm_ram[2]={7'd0, 5'd4, 5'd0, 3'b010, 5'd16,7'b0100011}; //    sw   x4, 16(x0)  // m[16] = x4
cm_ram[3]={12'd16,     5'd0, 3'b010, 5'd7, 7'b0000011}; //    lw   x7, 16(x0)  // x7 = m[16]
cm_ram[4]={12'd9,      5'd0, 3'b000, 5'd2, 7'b0010011}; //    addi x2, x0, 9   // x2 = 9
cm_ram[5]={7'd0, 5'd2, 5'd7, 3'b000, 5'd3, 7'b0110011}; //    add  x3, x7, x2  // x3 = x7 + x2
cm_ram[6]={7'd0, 5'd3, 5'd0, 3'b000, 5'd30,7'b0110011}; //    add  x30,x0, x3  // led = x3
cm_ram[7]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b1100011}; // L: beq  x0, x0, L
```

code171.v
program.txt



m_proc05 add, addi, lw, sw を処理するシングルサイクル版

code171.v
main15.xdc
50MHz clock freq.



m_proc05 シングルサイクル版

50MHz

WNS	TNS	LUT	FF	LUTRAM	BRAM
		2458	**66**	**2164**	**0**
3.413	0.000	3080	1215	2184	0

60MHz

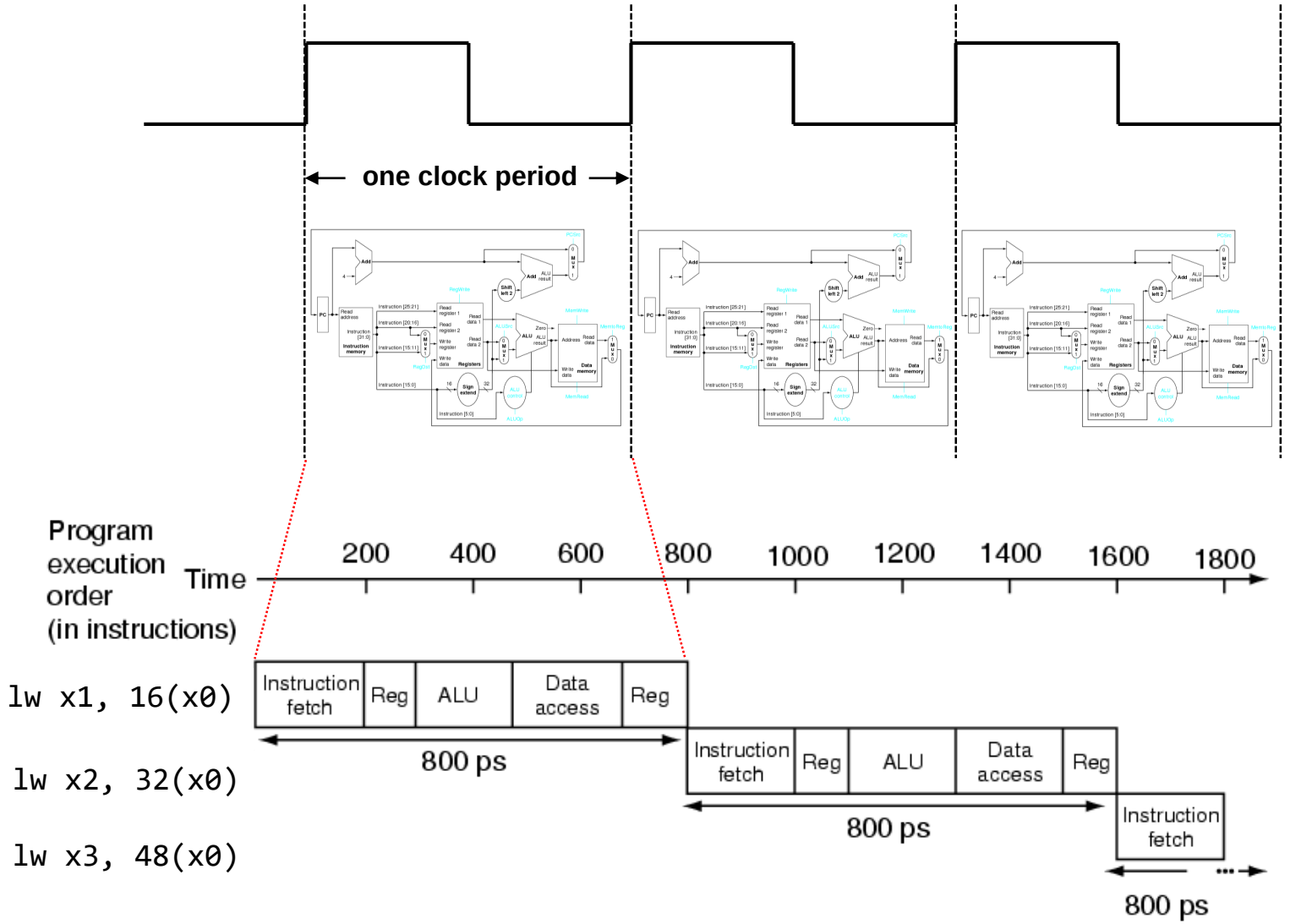
WNS	TNS	LUT	FF	LUTRAM	BRAM
		2458	**66**	**2164**	**0**
0.361	0.000	3081	1215	2184	0

70MHz

WNS	TNS	LUT	FF	LUTRAM	BRAM
		2458	**66**	**2164**	**0**
-0.023	-0.023	3081	1215	2184	0

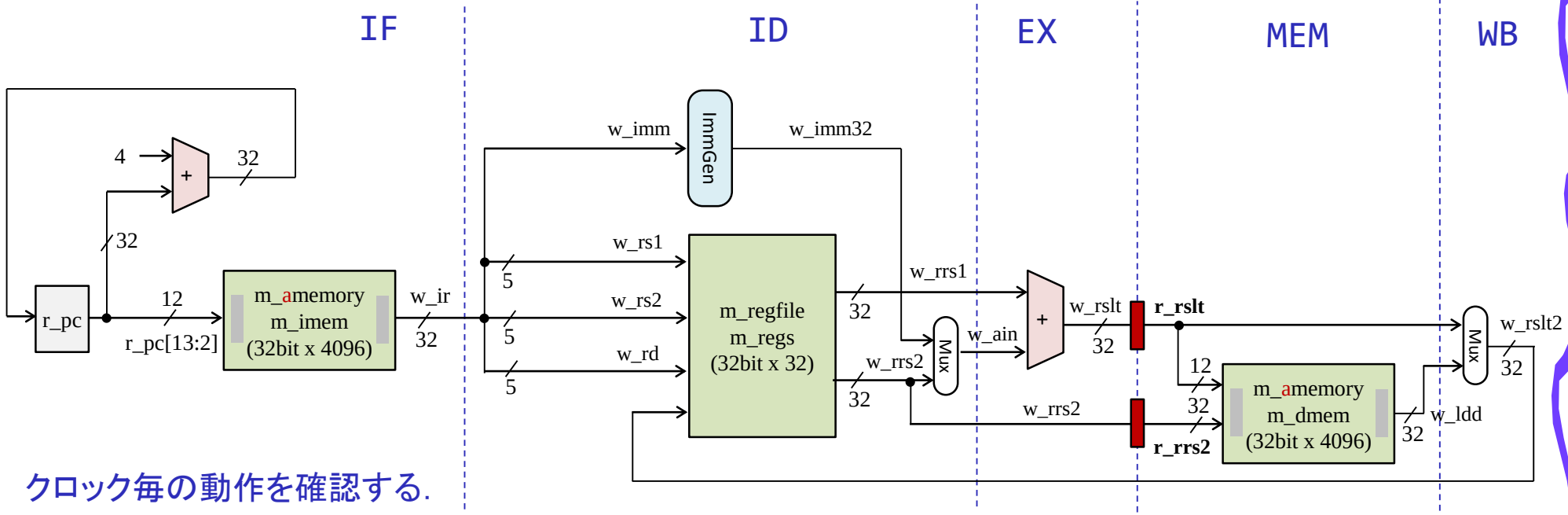
10MHz 単位で動作周波数を変化させて測定した場合の最高の動作周波数は 60MHz

Single Cycle Processor



m_proc08 (multi-cycle processor) マルチサイクル

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX) を1サイクル,
  メモリアクセス(MEM), ライトバック(WB) の処理を1サイクルで処理するマルチサイクルのプロセッサ
- 2サイクルで1命令を処理する IPC (instructions per cycle) = 0.5 のプロセッサ, add, addi, lw, sw をサポート



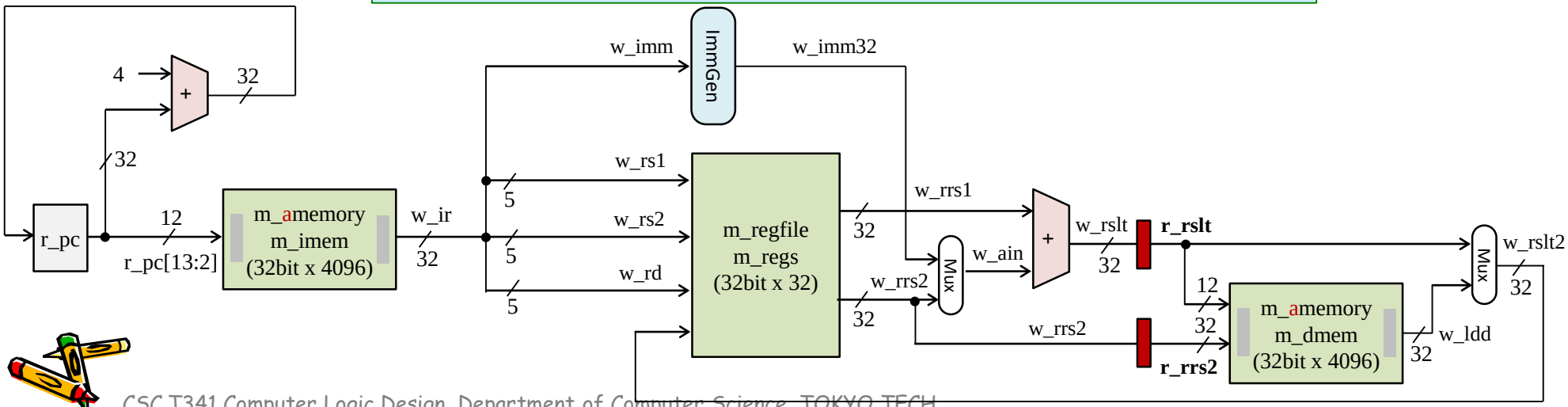
クロック毎の動作を確認する.

```
cm_ram[0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011}; //    add  x0, x0, x0  // NOP
cm_ram[1]={12'd55,     5'd0, 3'b000, 5'd4, 7'b0010011}; //    addi x4, x0, 55  // x4 = 55
cm_ram[2]={7'd0, 5'd4, 5'd0, 3'b010, 5'd16,7'b0100011}; //    sw   x4, 16(x0)  // m[16] = x4
cm_ram[3]={12'd16,     5'd0, 3'b010, 5'd7, 7'b0000011}; //    lw   x7, 16(x0)  // x7 = m[16]
cm_ram[4]={12'd9,      5'd0, 3'b000, 5'd2, 7'b0010011}; //    addi x2, x0, 9   // x2 = 9
cm_ram[5]={7'd0, 5'd2, 5'd7, 3'b000, 5'd3, 7'b0110011}; //    add  x3, x7, x2  // x3 = x7 + x2
cm_ram[6]={7'd0, 5'd3, 5'd0, 3'b000, 5'd30,7'b0110011}; //    add  x30,x0, x3  // led = x3
cm_ram[7]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b1100011}; // L: beq  x0, x0, L
```

m_proc08 (multi-cycle processor) マルチサイクル


```
cm_ram[0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011}; //    add  x0, x0, x0  // NOP
cm_ram[1]={12'd55,     5'd0, 3'b000, 5'd4, 7'b0010011}; //    addi x4, x0, 55  // x4 = 55
cm_ram[2]={7'd0, 5'd4, 5'd0, 3'b010, 5'd16,7'b0100011}; //    sw   x4, 16(x0)  // m[16] = x4
cm_ram[3]={12'd16,     5'd0, 3'b010, 5'd7, 7'b0000011}; //    lw   x7, 16(x0)  // x7 = m[16]
cm_ram[4]={12'd9,      5'd0, 3'b000, 5'd2, 7'b0010011}; //    addi x2, x0, 9   // x2 = 9
cm_ram[5]={7'd0, 5'd2, 5'd7, 3'b000, 5'd3, 7'b0110011}; //    add  x3, x7, x2  // x3 = x7 + x2
cm_ram[6]={7'd0, 5'd3, 5'd0, 3'b000, 5'd30,7'b0110011}; //    add  x30,x0, x3  // led = x3
cm_ram[7]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b1100011}; // L: beq  x0, x0, L
```

code172.v
program.txt

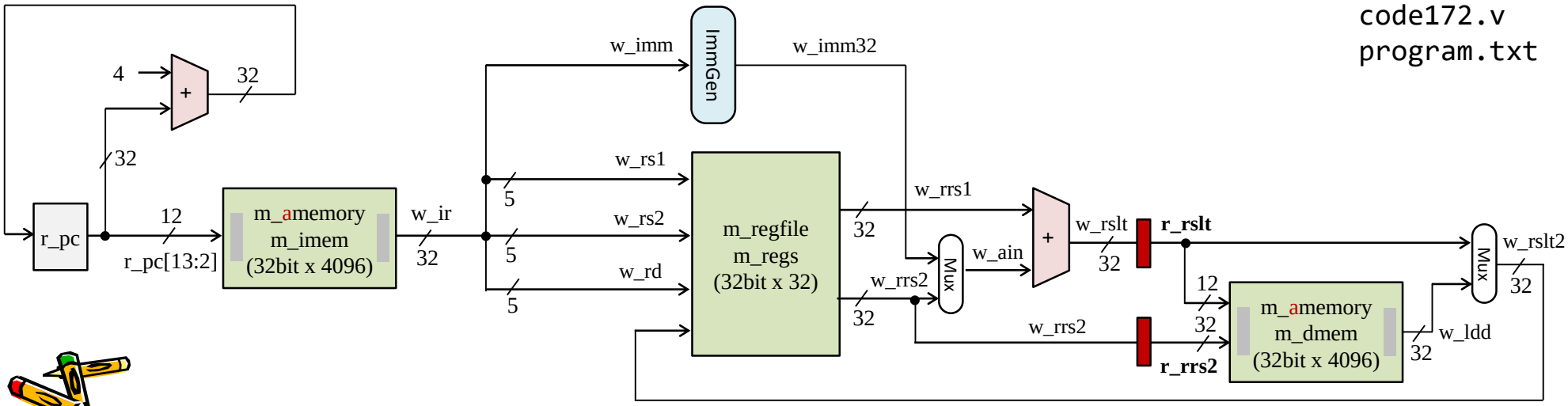


m_proc08 (multi-cycle processor) マルチサイクル

```
module m_proc08 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  wire [31:0] w_ir;
  wire [4:0]  w_op5 = w_ir[6:2];
  wire [4:0]  w_rs1 = w_ir[19:15];
  wire [4:0]  w_rs2 = w_ir[24:20];
  wire [4:0]  w_rd  = w_ir[11:7];

  reg [31:0] r_rslt, r_rrs2;

  reg r_state = 0;
  always @(posedge w_clk) #5
    if(w_ce) r_state <= r_state + 1;
```

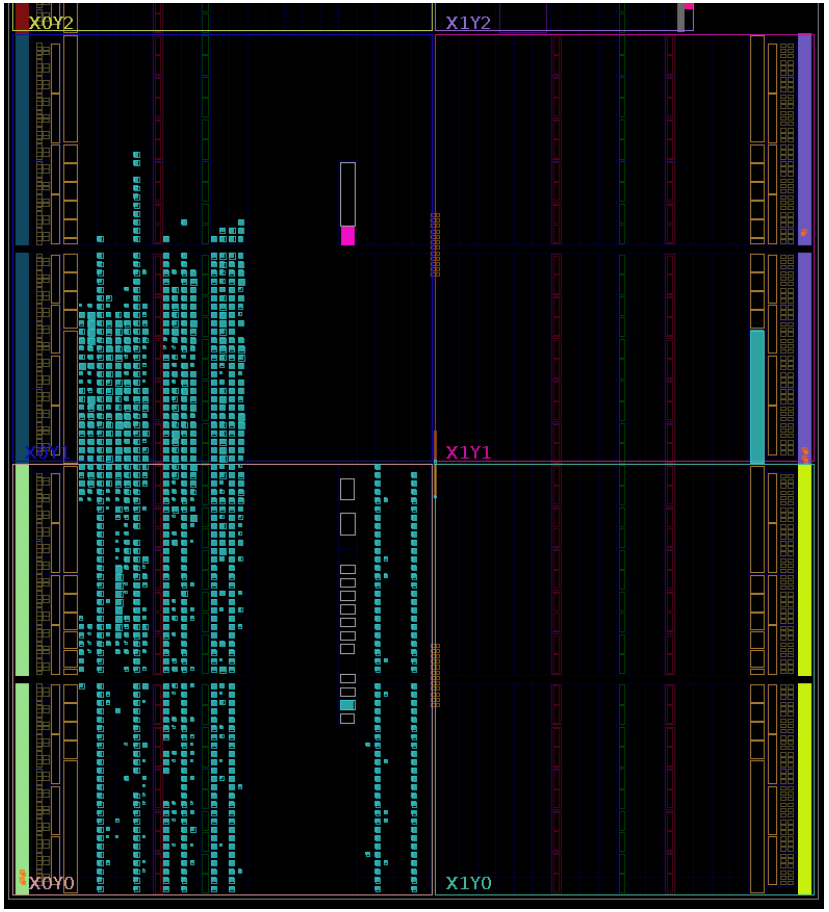
```
  wire #4 w_we = w_ce & (w_op5==5'b01100 || w_op5==5'b00100 || w_op5==5'b00000);

  wire [31:0] w_imm, w_rrs1, w_rrs2, w_ain, w_rslt, w_ldd, w_rslt2;

  always @(posedge w_clk) if(w_ce & r_state==0) r_rslt <= w_rslt;
  always @(posedge w_clk) if(w_ce & r_state==0) r_rrs2 <= w_rrs2;
  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);
  m_immgen m_immgen0 (w_ir, w_imm);
  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_we & r_state, w_rslt2, w_rrs1, w_rrs2);
  assign #3 w_ain = (w_op5==5'b01100) ? w_rrs2 : w_imm;
  assign #9 w_rslt = w_rrs1 + w_ain;
  m_amemory m_dmem (w_clk, r_rslt[13:2], (w_op5==5'b01000 & r_state), r_rrs2, w_ldd);
  assign #3 w_rslt2 = (w_op5==5'b00000) ? w_ldd : r_rslt;
  always @(posedge w_clk) #5 if(w_ce & r_state & r_pc!=24) r_pc <= r_pc + 4;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_we & r_state & w_rd==30) r_led <= w_rslt2;
  assign w_led = r_led;
endmodule
```

code172.v
program.txt



m_proc08 (multi-cycle processor) マルチサイクル

code172.v (m_proc08)
main15.xdc
80MHz clock freq.



m_proc05 シングルサイクル版　**Perf = 60 x 1.0 = 60.0**

60MHz

WNS	TNS	LUT	FF	LUTRAM	BRAM
		2458	**66**	**2164**	**0**
0.361	0.000	3081	1215	2184	0

70MHz

WNS	TNS	LUT	FF	LUTRAM	BRAM
		2458	**66**	**2164**	**0**
-0.023	-0.023	3081	1215	2184	0

m_proc08 2段のマルチサイクル版　**Perf = 110 x 0.5 = 55.0**

90MHz

WNS	TNS	LUT	FF	LUTRAM	BRAM
		2461	**219**	**2164**	**0**
1.306	0.000	3084	1368	2184	0

100MHz

WNS	TNS	LUT	FF	LUTRAM	BRAM
		2461	**219**	**2164**	**0**
0.284	0.000	3083	1368	2184	0

110MHz

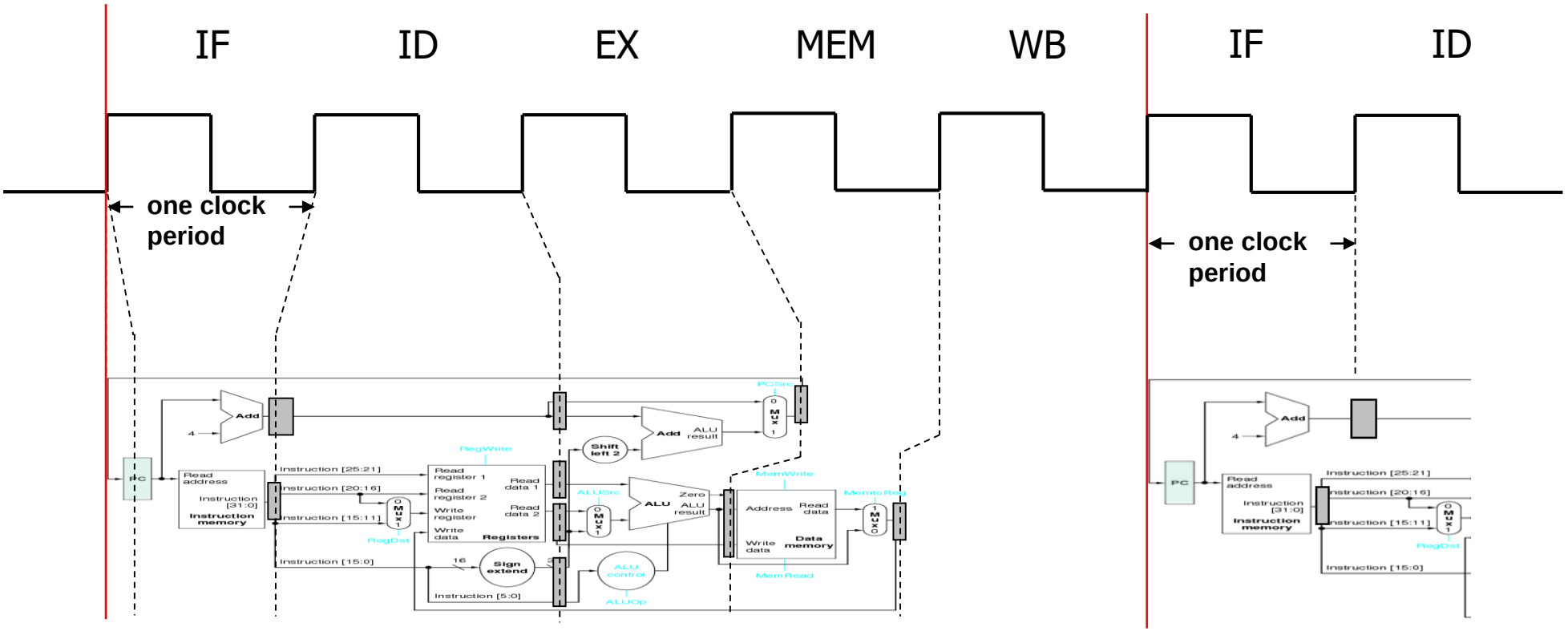
WNS	TNS	LUT	FF	LUTRAM	BRAM
		2461	**219**	**2164**	**0**
0.229	0.000	3084	1368	2184	0

120MHz

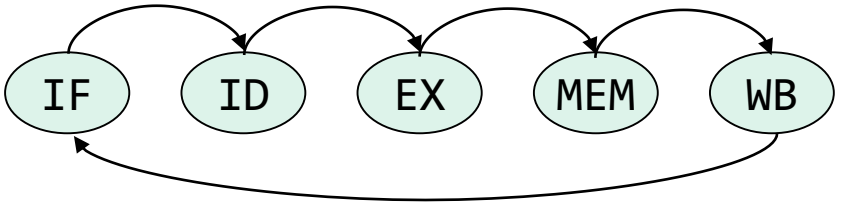
WNS	TNS	LUT	FF	LUTRAM	BRAM
		2461	**219**	**2164**	**0**
-0.027	-0.071	3091	1368	2184	0

10MHz単位で動作周波数を変化させて測定した場合の最高の動作周波数は 110MHz

マルチサイクルのプロセッサ Multi-cycle Processor



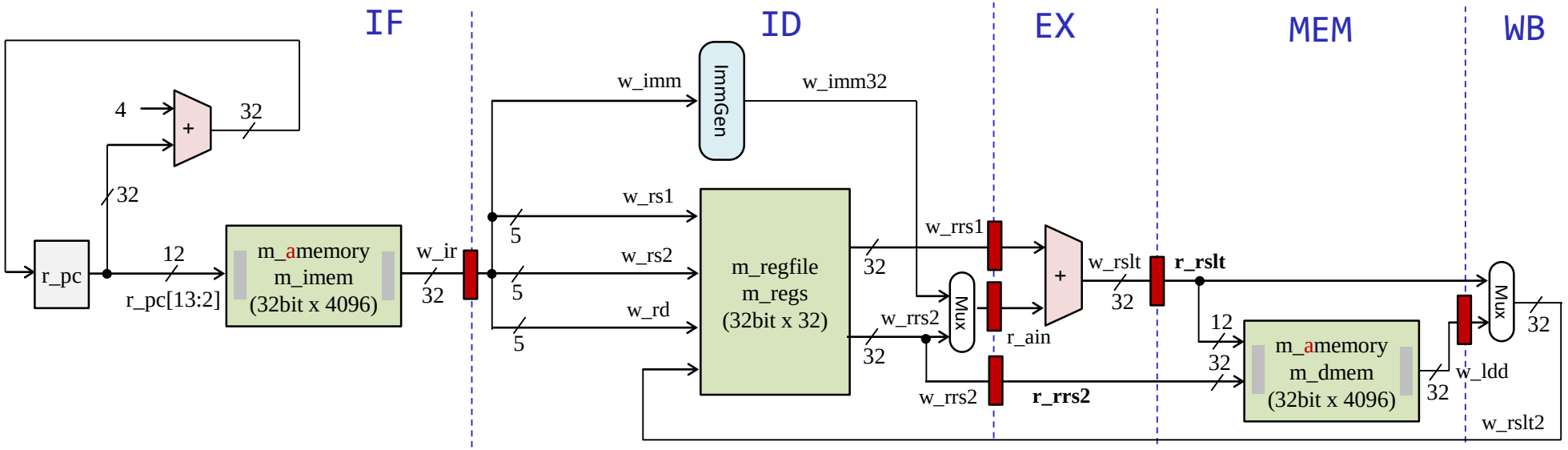
State Machine Diagram



0.1 x 3 + 0.1 x 5 + 0.8 x 4 = 4.0

m_proc09 (multi-cycle processor) マルチサイクル

- 5サイクルで1命令を処理する IPC = 0.2 のマルチサイクルのプロセッサ, add, addi, lw, sw をサポート
- 最も遅延の長い(長い時間を要する)ステップがプロセッサの動作周波数を決める.

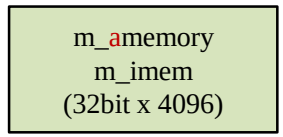


クロック毎の動作を確認する.

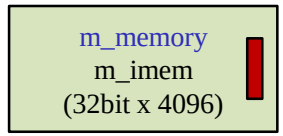
```
cm_ram[0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011}; //    add  x0, x0, x0  // NOP
cm_ram[1]={12'd55,     5'd0, 3'b000, 5'd4, 7'b0010011}; //    addi x4, x0, 55  // x4 = 55
cm_ram[2]={7'd0, 5'd4, 5'd0, 3'b010, 5'd16,7'b0100011}; //    sw   x4, 16(x0)  // m[16] = x4
cm_ram[3]={12'd16,     5'd0, 3'b010, 5'd7, 7'b0000011}; //    lw   x7, 16(x0)  // x7 = m[16]
cm_ram[4]={12'd9,      5'd0, 3'b000, 5'd2, 7'b0010011}; //    addi x2, x0, 9   // x2 = 9
cm_ram[5]={7'd0, 5'd2, 5'd7, 3'b000, 5'd3, 7'b0110011}; //    add  x3, x7, x2  // x3 = x7 + x2
cm_ram[6]={7'd0, 5'd3, 5'd0, 3'b000, 5'd30,7'b0110011}; //    add  x30,x0, x3  // led = x3
cm_ram[7]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b1100011}; // L: beq  x0, x0, L
```

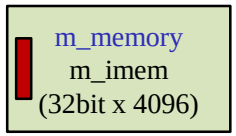
非同期(asynchronous)メモリと同期(synchronous)メモリ

```
module m_amemory (w_clk, w_addr, w_we, w_din, w_dout); // asynchronous memory
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output wire [31:0] w_dout;
  reg [31:0]         cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) if (w_we) cm_ram[w_addr] <= w_din;
  assign #20 w_dout = cm_ram[w_addr];
`include "program.txt"
endmodule

module m_memory (w_clk, w_addr, w_we, w_din, r_dout); // synchronous memory
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output reg  [31:0] r_dout;
  reg [31:0]         cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) if (w_we) cm_ram[w_addr] <= w_din;
  always @(posedge w_clk) r_dout <= cm_ram[w_addr];
`include "program.txt"
endmodule
```

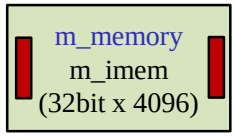


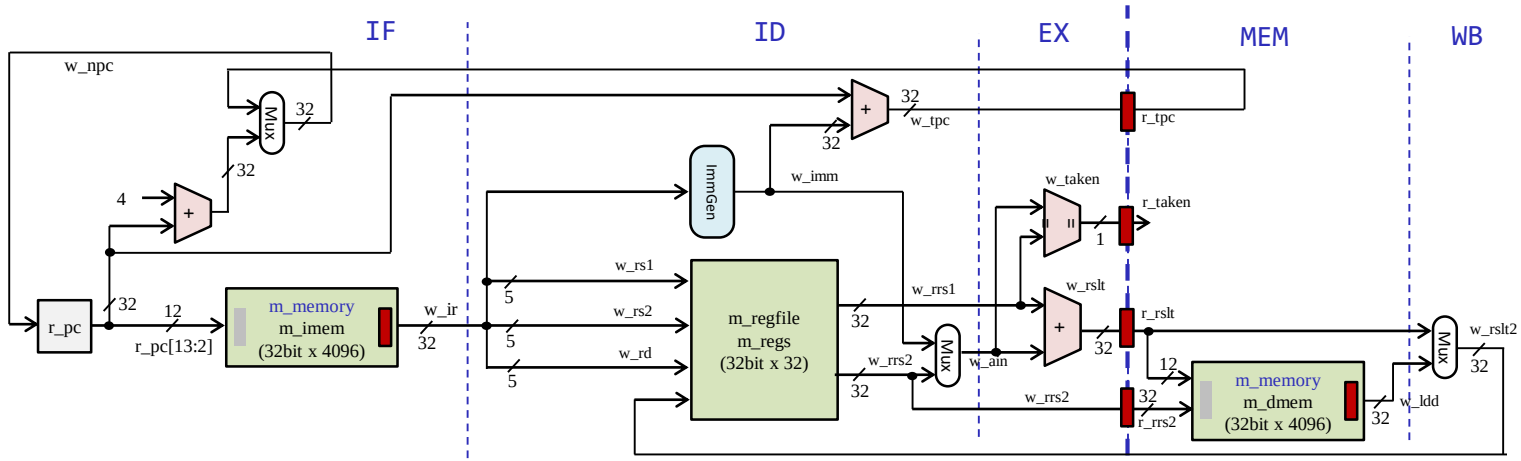
非同期メモリ





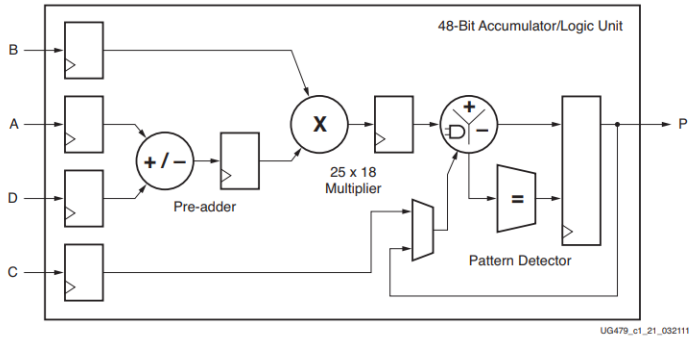
同期メモリ



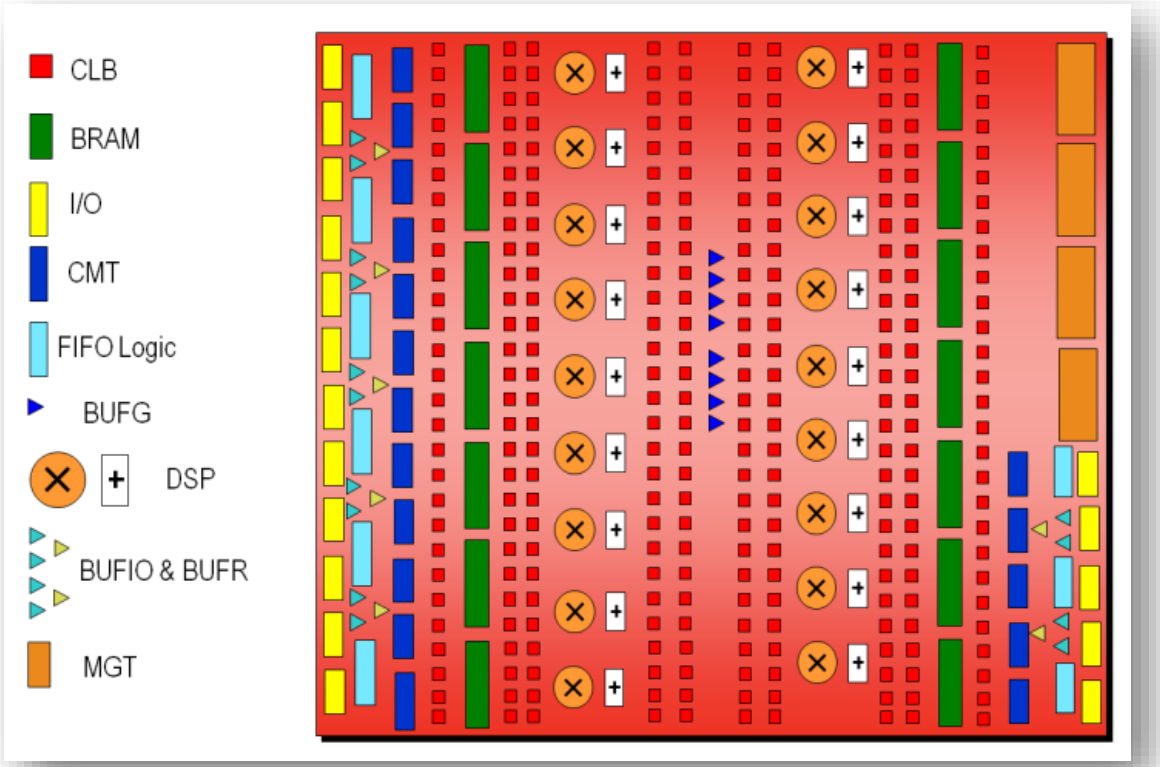


Artix-7 Architecture Overview

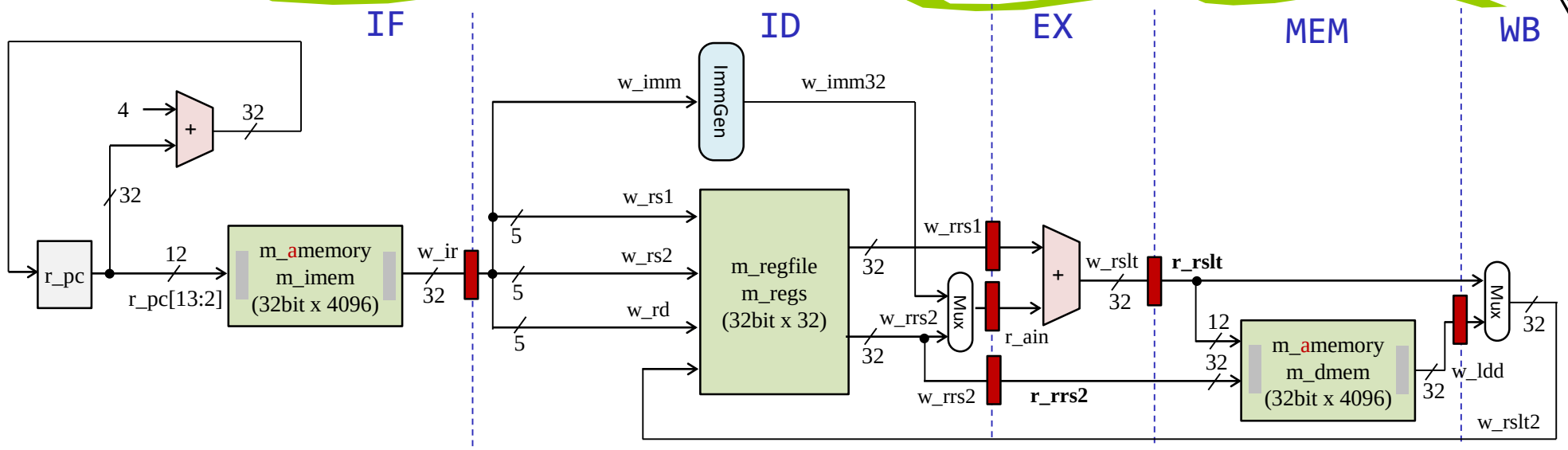
- CLB (Configurable Logic Block)
- BRAM (Block RAM, embedded memory, synchronous memory)
- DSP (Digital Signal Processing)
- CMT (Clock Management Tile)
- Routing fabric



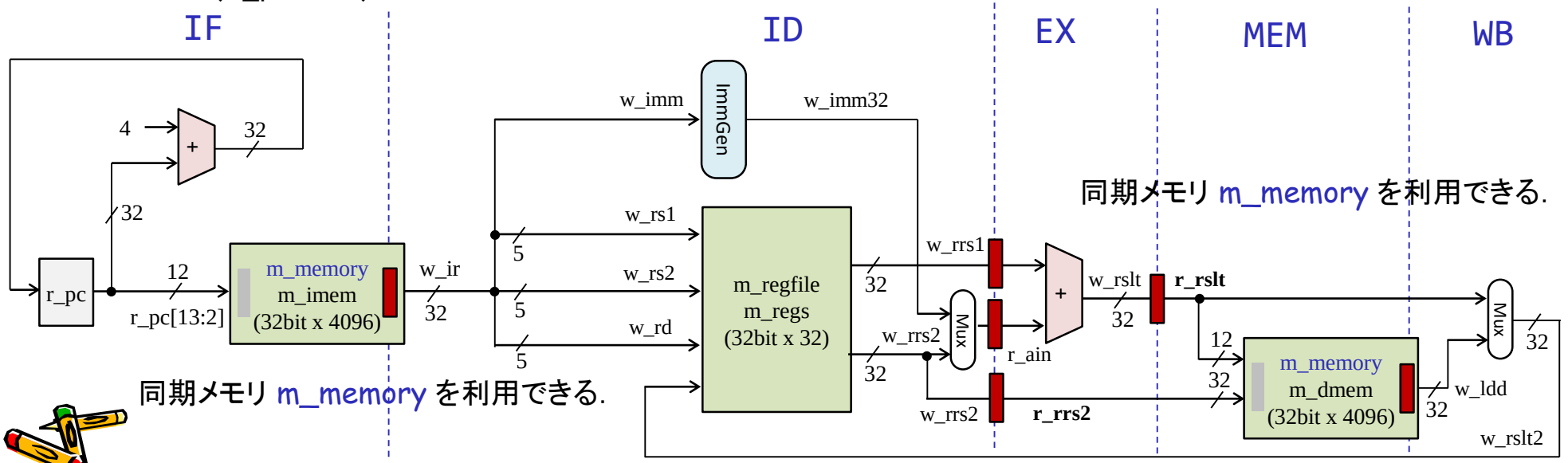
DSP Slice



m_proc10 (multi-cycle processor) マルチサイクル



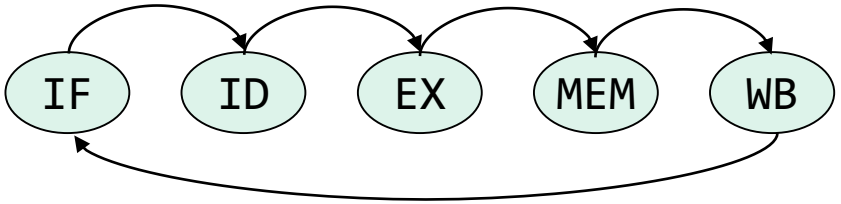
code174.v (m_proc10)



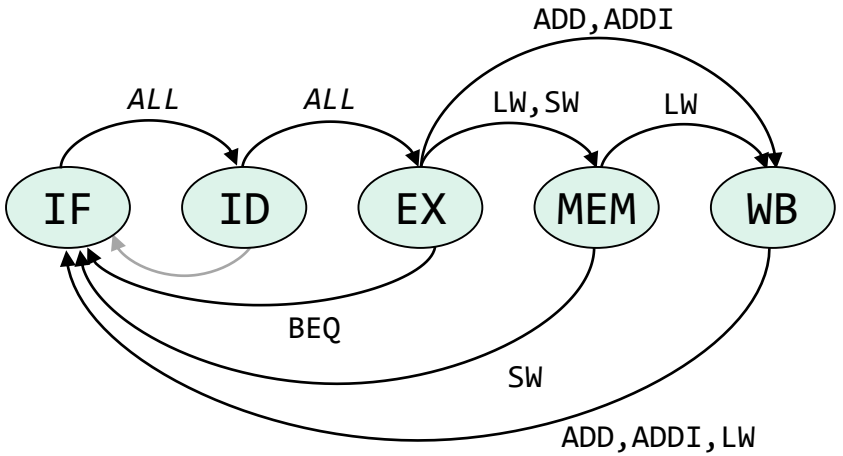
同期メモリ m_memory を利用できる.

同期メモリ m_memory を利用できる.

Optimization for Multi-cycle Processors



Simple State Machine Diagram

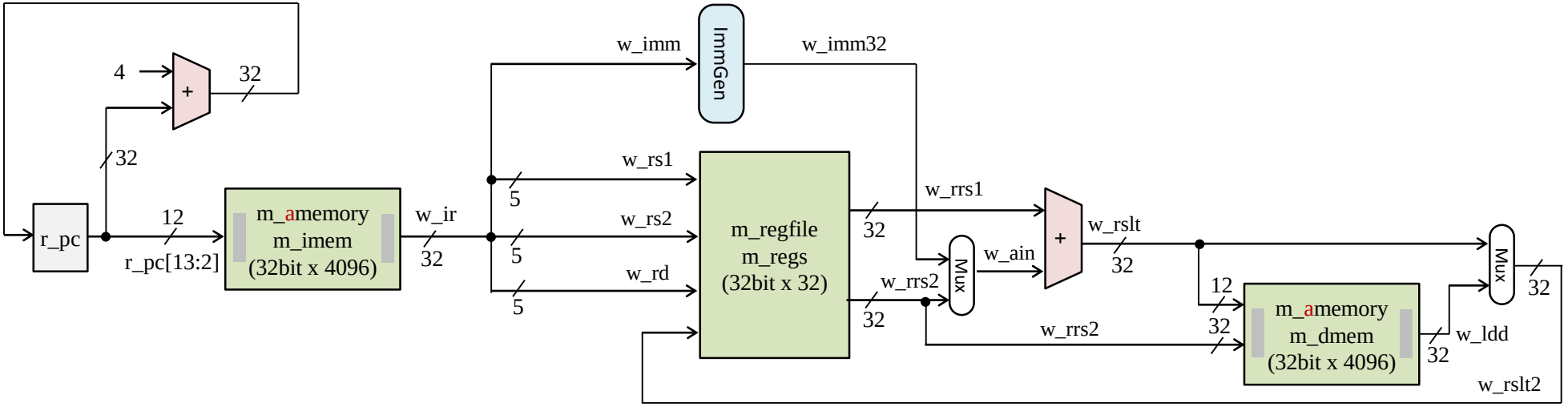


Optimized State Machine Diagram

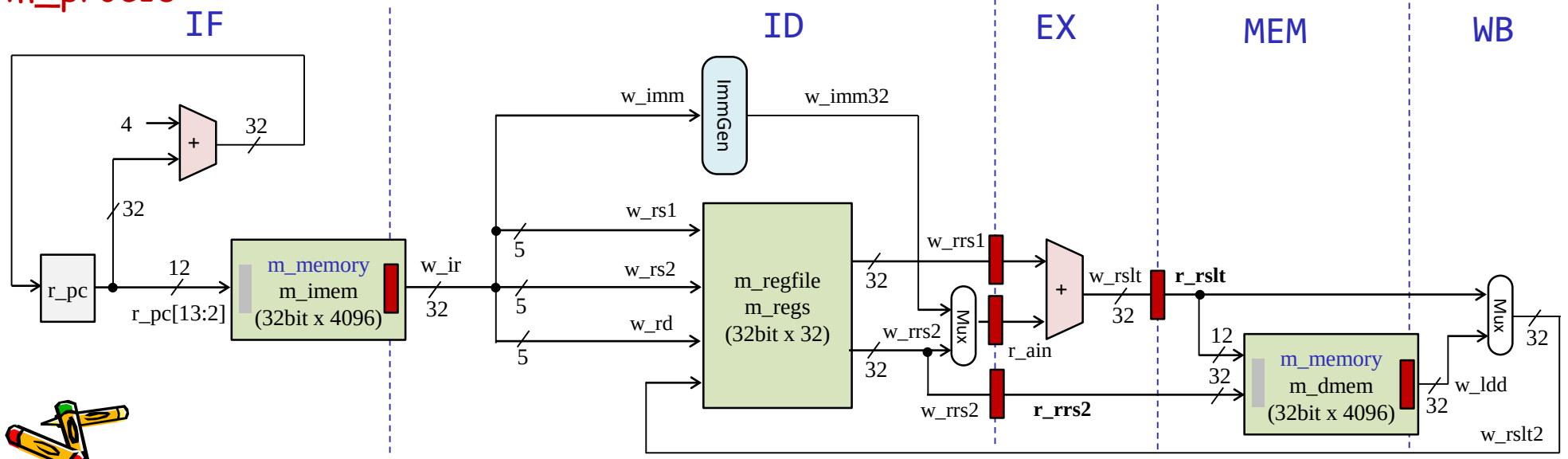
$0.1 \times 3 + 0.1 \times 5 + 0.8 \times 4 = 4.0$

Comparison of the two processor architectures

m_proc05 (code171.v)



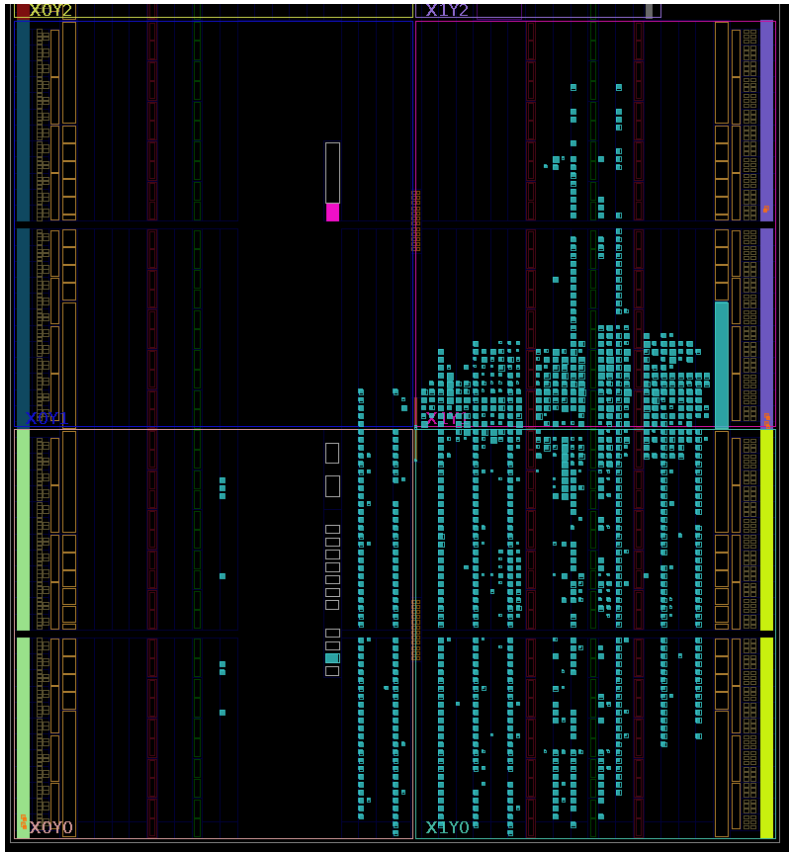
m_proc10

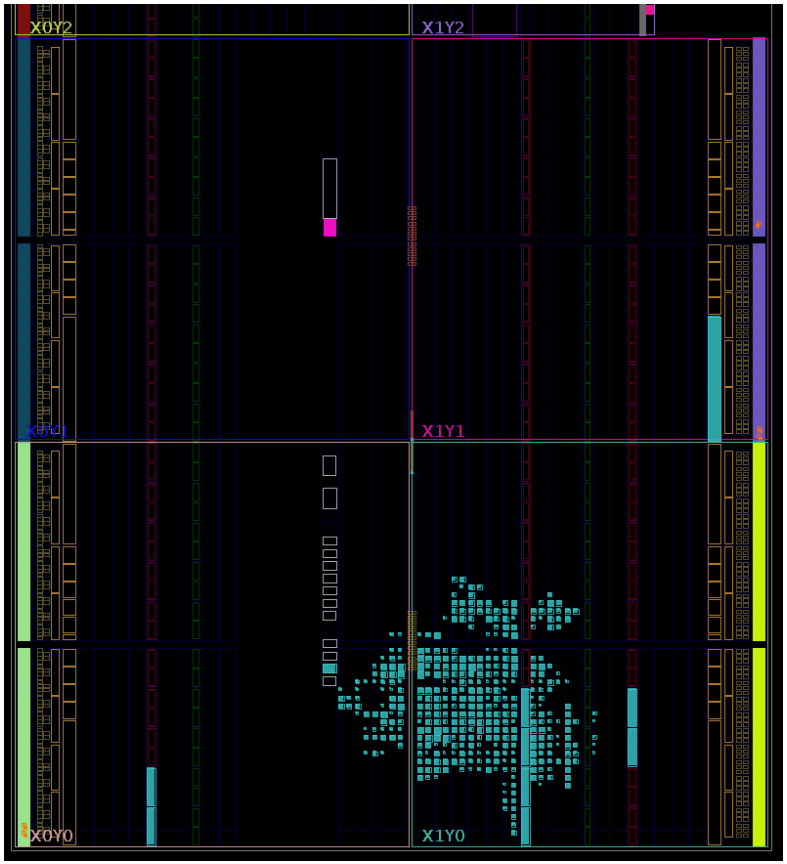


m_proc05 add, addi, lw, sw を処理するシングルサイクル版

code171.v
main15.xdc
50MHz clock freq.

左は，非同期メモリを使うようにm_proc05 を記述して分散メモリとなる場合
右は，同期メモリを使うようにm_proc05 を記述して BRAM となる場合

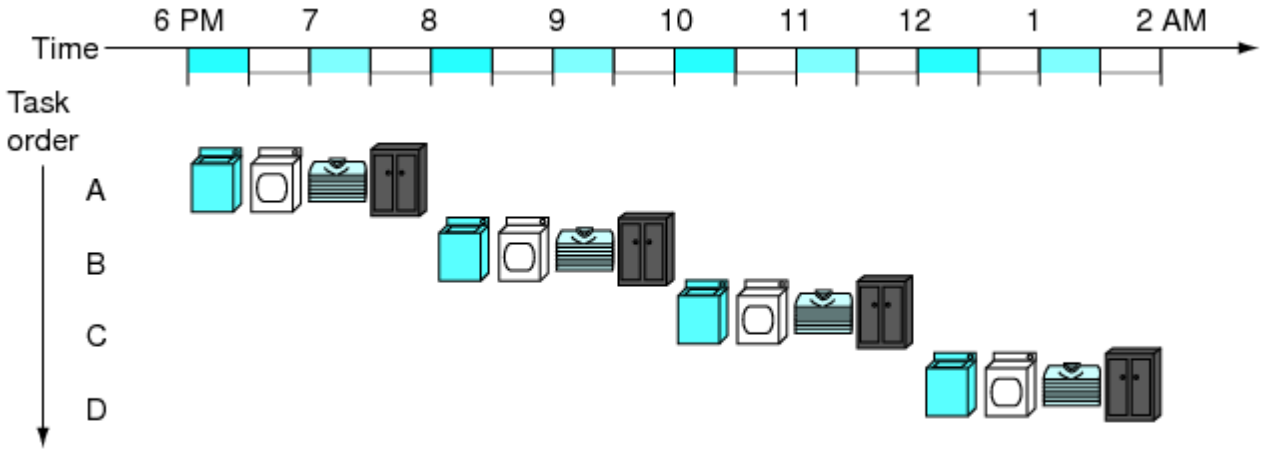




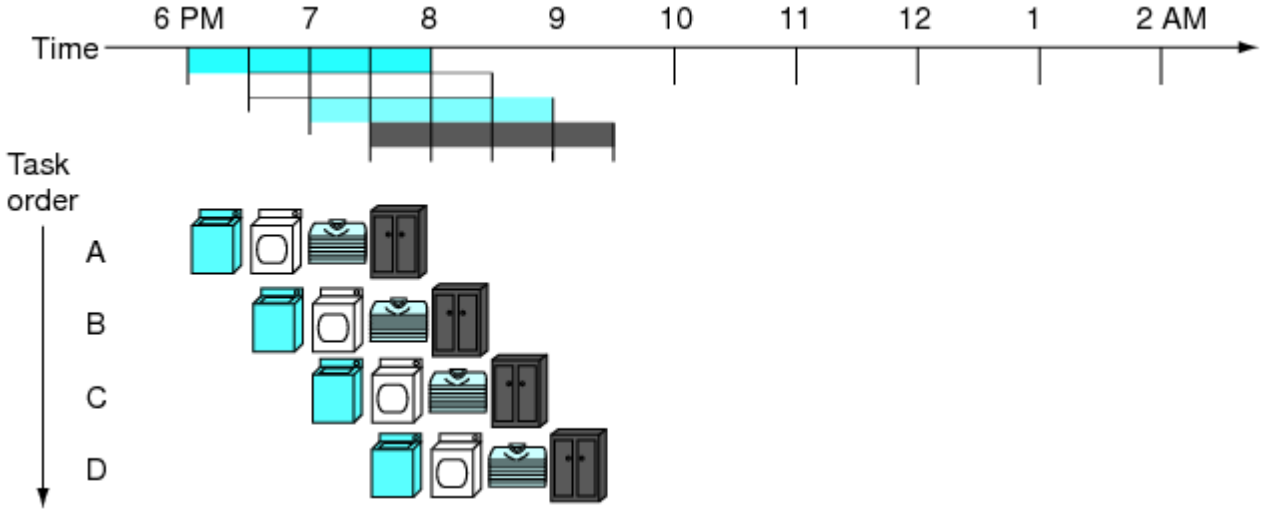
code171.v の命令メモリとデータメモリを m_memory に変更した版

An overview of pipelining (topic of the next lecture)

- Non pipelining (Multi-cycle)

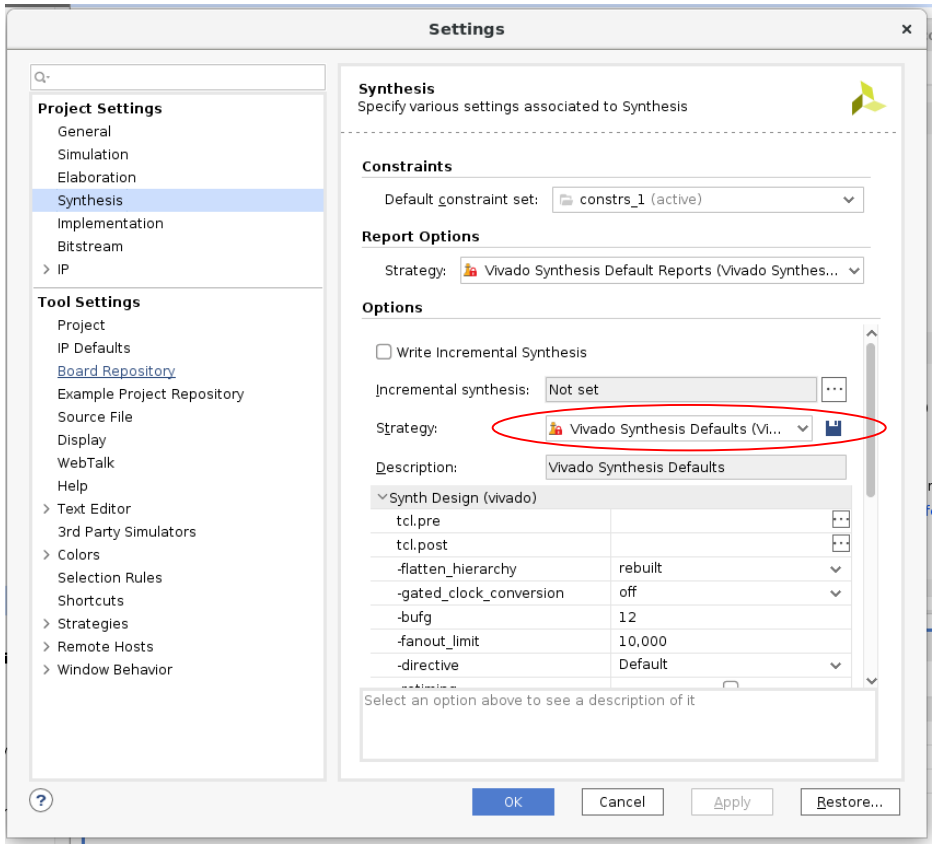


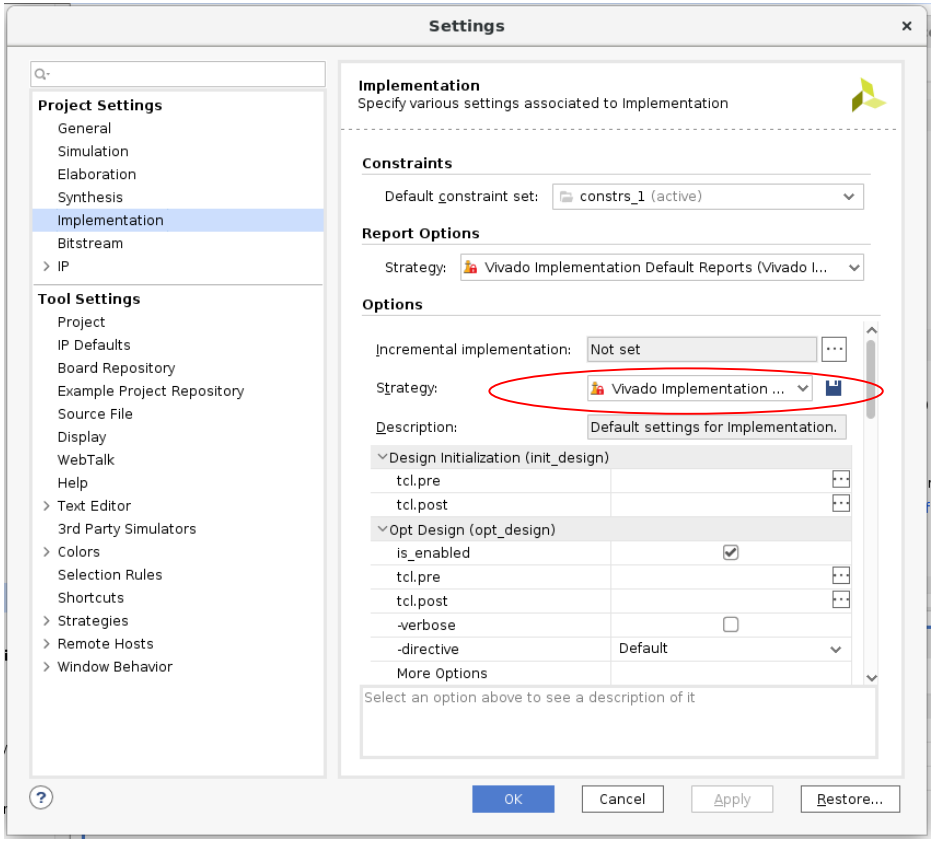
- Pipelining



補足：Vivado の最適化オプション

- Settings -> Synthesis -> Strategy : Vivado Synthesis Defaults
- Settings -> Implementation -> Strategy: Vivado Implementation Defaults





補足：Vivado の最適化オプション

- プロセッサの高性能化のための推奨オプションは次の通り．デザインコンテストでは，これらのVivadoのオプションを適切なものに変更してもよい．

- Settings -> Synthesis -> Strategy :
  Flow_PerfOptimized_high
- Settings -> Implementation -> Strategy:
  Performance_ExplorePostRoutePhysOpt

- ただし，これらの最適化オプションを変更すると，論理合成と配置配線の時間が長くなるので，適切に使い分ける必要がある．

References

- Computer Logic Design support page
  - https://www.arch.cs.titech.ac.jp/lecture/CLD/
- ACRi Room
  - https://gw.acri.c.titech.ac.jp
- ACRi Blog
  - https://www.acri.c.titech.ac.jp/wordpress/
- 情報工学系計算機室
  - http://www.csc.titech.ac.jp/
- Xilinx Vivado Design Suite
  - https://japan.xilinx.com/products/design-tools/vivado.html
- Digilent Arty A7-35T
  - https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start
- Verilog HDL
  - https://ja.wikipedia.org/wiki/Verilog