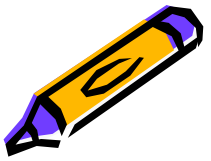
2023年度（令和5年）版

Ver. 2023-10-24a

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

7. パイプラインプロセッサ (2)
Pipelined Processor (2)



www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

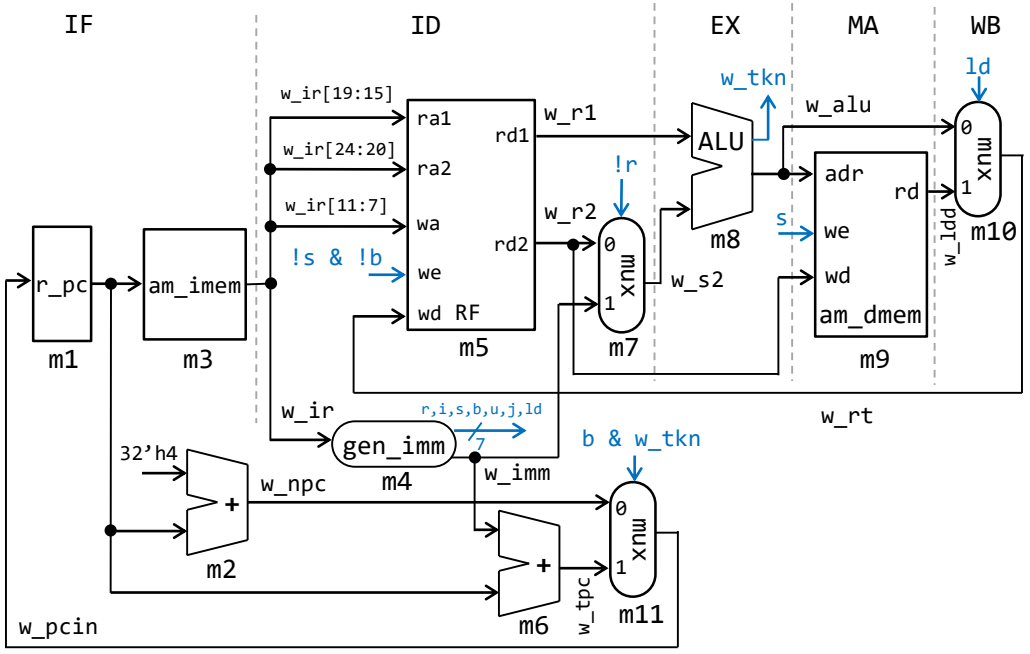
吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

プロセッサが命令を処理するための5つのステップ

- 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.
- EX (Execution)
  命令操作の実行またはアドレスの生成を行う.
- MA (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.

proc5s: single-cycle processor

- supporting add, addi, lw, sw, bne instructions

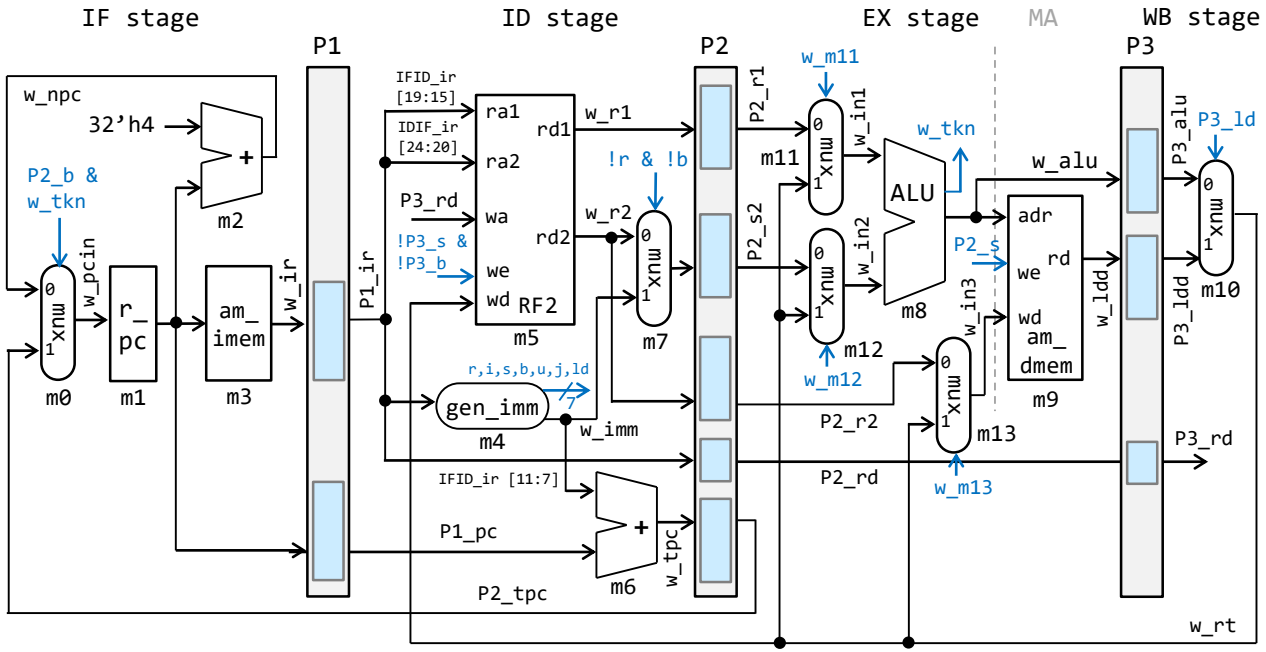



```
module m_proc5s(w_clk);
  input wire w_clk;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin;
  wire w_tkn;
  reg [31:0] r_pc=0; // m1
  assign w_pcin = (w_b & w_tkn) ?  w_tpc : w_npc; // m11
  assign w_npc  = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld;
  m_gen_imm m4 (w_ir, w_imm, w_r, w_i,
                w_s, w_b, w_u, w_j, w_ld);
  m_RF m5 (w_clk, w_ir[19:15], w_ir[24:20], w_r1, w_r2,
           w_ir[11:7], !w_s & !w_b, w_rt);
  assign w_tpc = r_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  m_alu m8 (w_r1, w_s2, w_alu, w_tkn);
  m_am_dmem m9 (w_clk, w_alu, w_s, w_r2, w_ldd);
  assign w_rt = (w_ld) ? w_ldd : w_alu; // m10
  always @(posedge w_clk) r_pc <= w_pcin;
endmodule
```

プロセッサが命令を処理するための5つのステップ

- 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.
- EX (Execution)
  命令操作の実行またはアドレスの生成を行う.
- MA (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.

proc8: 4-stage pipelining processor



Processor simulation

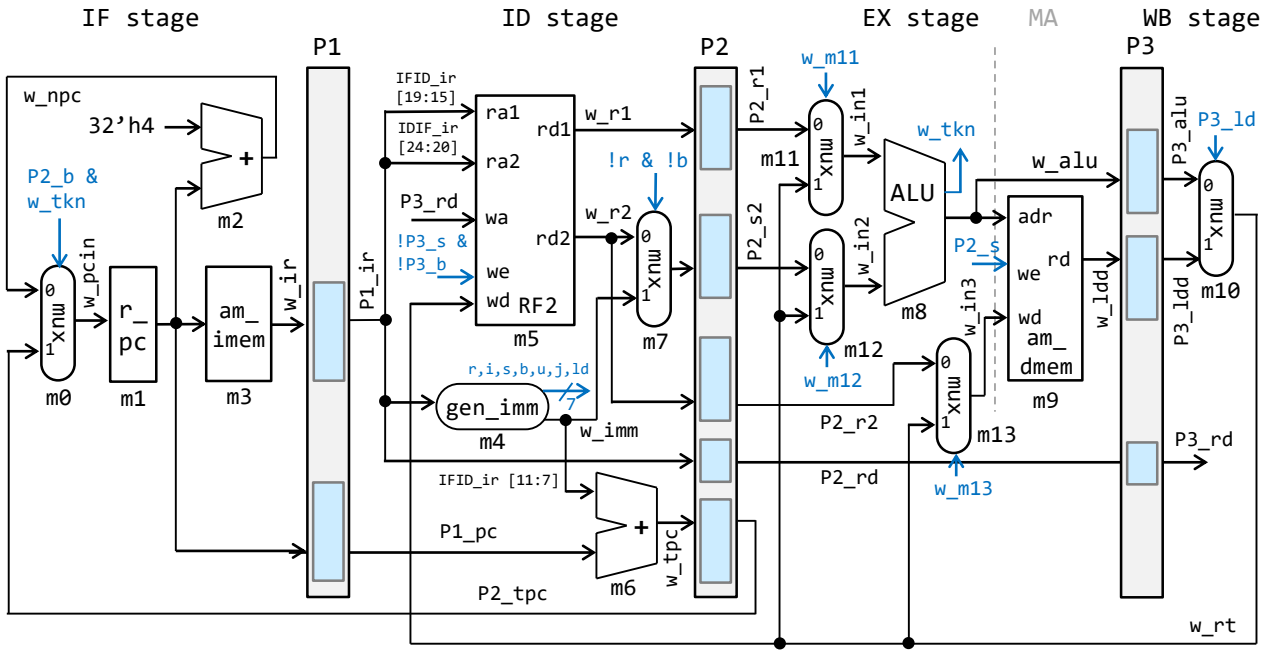
```
// 00    addi x1,x0,3
// 04    add  x2,x1,x1
// 08    addi x10,x2,5
// 0c    HALT
```

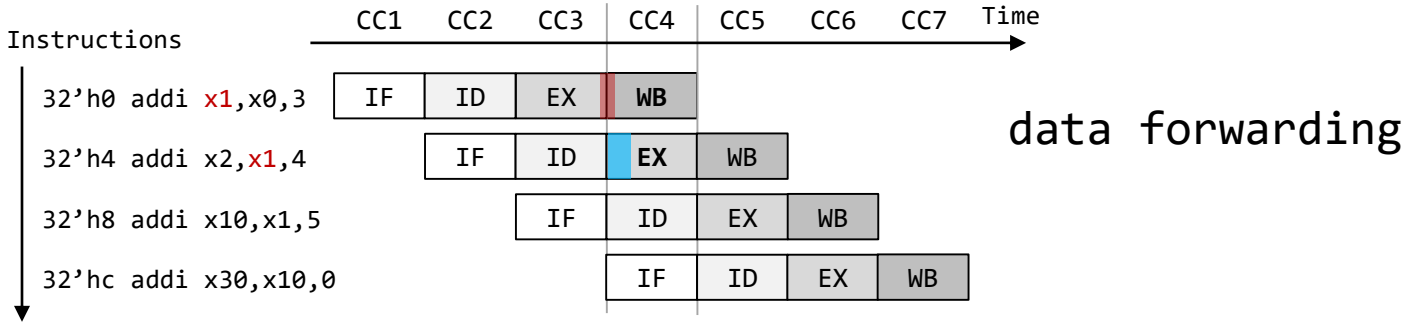
```
$ iverilog -DP1 proc8s.v
$ ./a.out

CC01 00000000 00000000 00000000 00000000          0          0          0
CC02 00000004 00000000 00000000 00000000          0          0          0
CC03 00000008 00000004 00000000 00000000          0          3          3
CC04 0000000c 00000008 00000004 00000000          3          3          6
CC05 00000010 0000000c 00000008 00000004          6          5         11
CC06 00000014 00000010 0000000c 00000008         11          0         11
CC07 00000018 00000014 00000010 0000000c          0          0          0
```

```
module m_sim(w_clk, w_cc); /* please wrap me by m_top_wrapper */
  input wire w_clk; input wire [31:0] w_cc;
  m_proc8s m (w_clk);
  initial begin
    `define MM m.m3.mem
    `include "asm.txt"
  end
  initial #99 forever #100 $display("CC%02d %h %h %h %h %d %d %d",
    w_cc, m.r_pc, m.P1_pc, m.P2_pc, m.P3_pc,
    m.w_in1, m.w_in2, m.w_alu);
endmodule
```

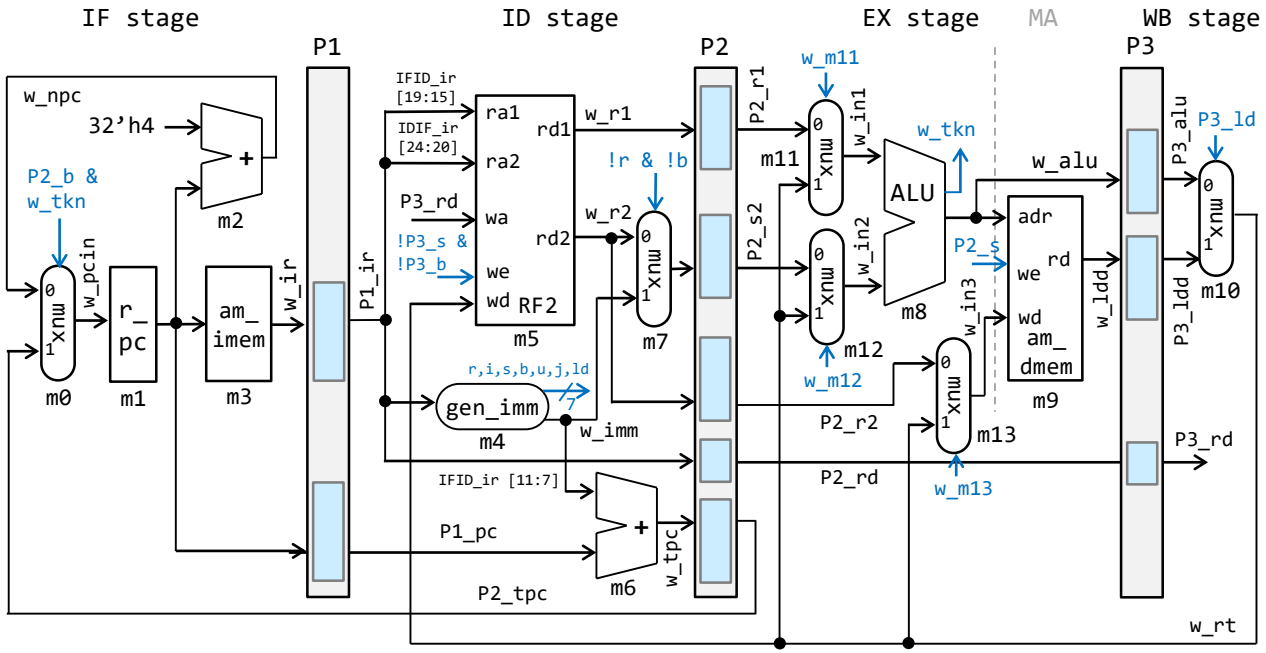
proc8: 4-stage pipelining processor

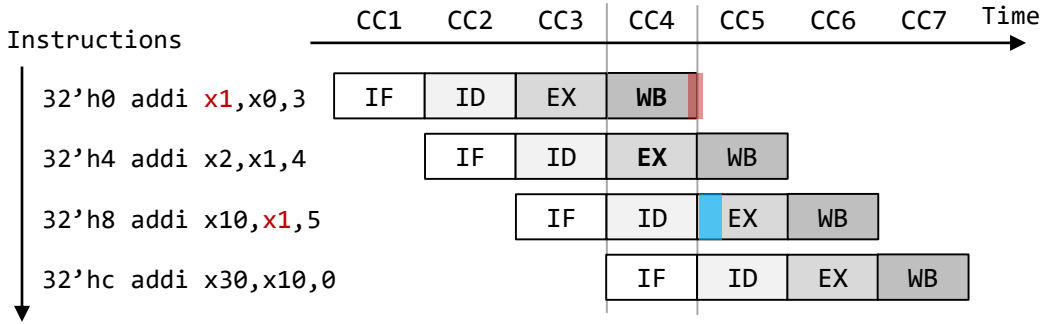




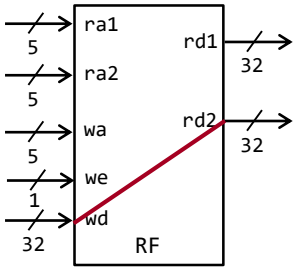
data forwarding

proc8: 4-stage pipelining processor

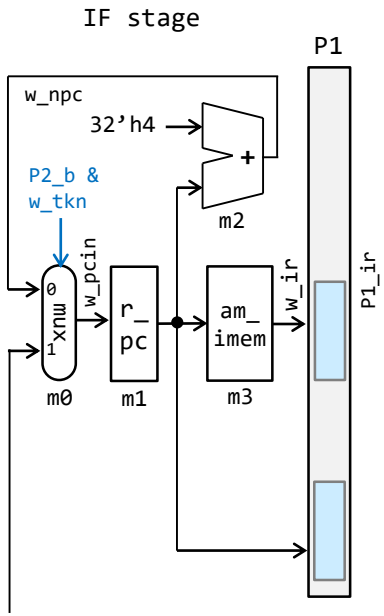




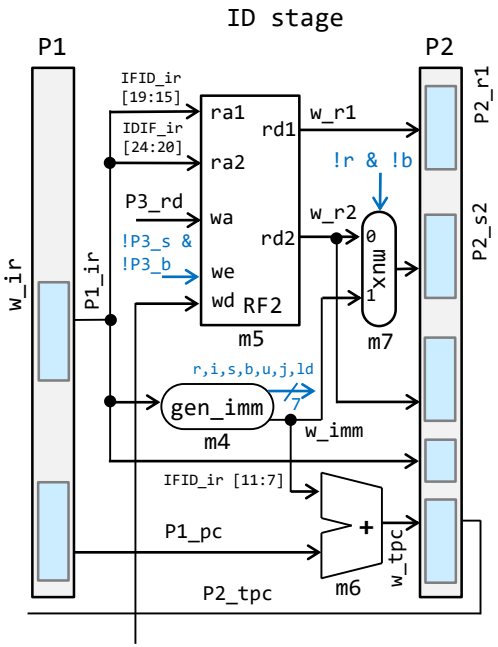
RF2 (Register File) with bypassing



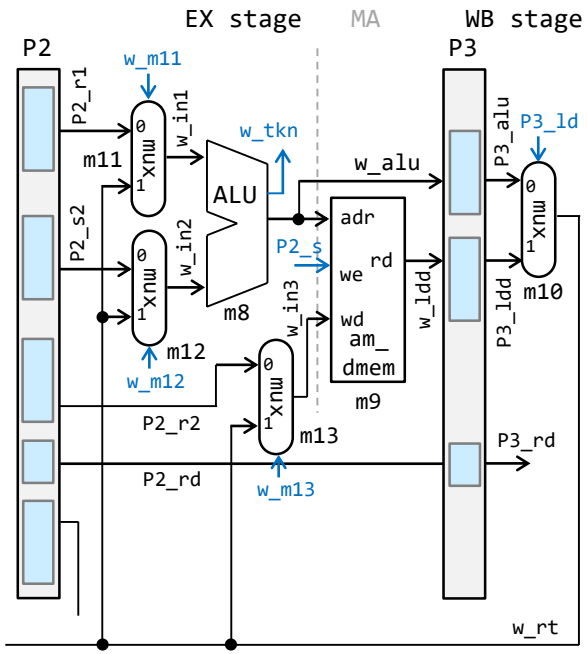

```
module m_RF2(w_clk, w_radr1, w_radr2, w_rd1, w_rd2, w_wadr, w_we, w_wd);
  input  wire w_clk, w_we;
  input  wire [4:0] w_radr1, w_radr2, w_wadr;
  output wire [31:0] w_rd1, w_rd2;
  input  wire [31:0] w_wd;
  reg [31:0] mem [0:31];
  wire w_bp1 = (w_we & w_radr1==w_wadr);
  wire w_bp2 = (w_we & w_radr2==w_wadr);
  assign w_rd1 = (w_radr1==5'd0) ? 32'd0 : (w_bp1) ? w_wd : mem[w_radr1];
  assign w_rd2 = (w_radr2==5'd0) ? 32'd0 : (w_bp2) ? w_wd : mem[w_radr2];
  always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
  always @(posedge w_clk) if (w_we & w_wadr==5'd30) $finish;
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 32'd0;
endmodule
```




```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  reg P1_v=0, P2_v=0, P3_v=0;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin, w_in1, w_in2, w_in3;
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld, w_tkn;
  reg [31:0] r_pc = 0; // m1
  wire w_miss = P2_b & w_tkn & P2_v;
  assign w_pcin = (w_miss) ? P2_tpc : w_npc; // m0
  assign w_npc = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  m_gen_imm m4 (P1_ir, w_imm, w_r, w_i, w_s, w_b, w_u, w_j, w_ld);
  m_RF2 m5 (w_clk, P1_ir[19:15], P1_ir[24:20], w_r1, w_r2,
            P3_rd, !P3_s & !P3_b & P3_v, w_rt);
  assign w_tpc = P1_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  always @(posedge w_clk) begin
    {P1_v, P2_v, P3_v} <= {!w_miss, !w_miss & P1_v, P2_v};
    {r_pc, P1_ir, P1_pc, P2_pc} <= {w_pcin, w_ir, r_pc, P1_pc};
    {P2_r1, P2_r2, P2_s2, P2_tpc} <= {w_r1, w_r2, w_s2, w_tpc};
    {P2_r, P2_s, P2_b, P2_ld} <= {w_r, w_s, w_b, w_ld};
    {P2_rs2, P2_rs1, P2_rd} <= {P1_ir[24:15], P1_ir[11:7]};
    {P3_pc, P3_ld} <= {P2_pc, P2_ld};
    {P3_alu, P3_ldd, P3_rd} <= {w_alu, w_ldd, P2_rd};
  end
  assign w_alu = w_in1 + w_in2;   // m8
  assign w_tkn = w_in1 != w_in2;  // m8
  m_am_dmem m9 (w_clk, w_alu, P2_s & P2_v, w_in3, w_ldd);
  assign w_rt = (P3_ld) ? P3_ldd : P3_alu; // m10
  wire w_f =!P3_s & !P3_b & |P3_rd & P3_v;
  assign w_in1 = (w_f & P2_rs1==P3_rd) ? w_rt : P2_r1; // m11
  assign w_in3 = (w_f & P2_rs2==P3_rd) ? w_rt : P2_r2; // m13
  assign w_in2 = (w_f & P2_rs2==P3_rd & (P2_r|P2_b)) ? w_rt : P2_s2;// m12
endmodule
```

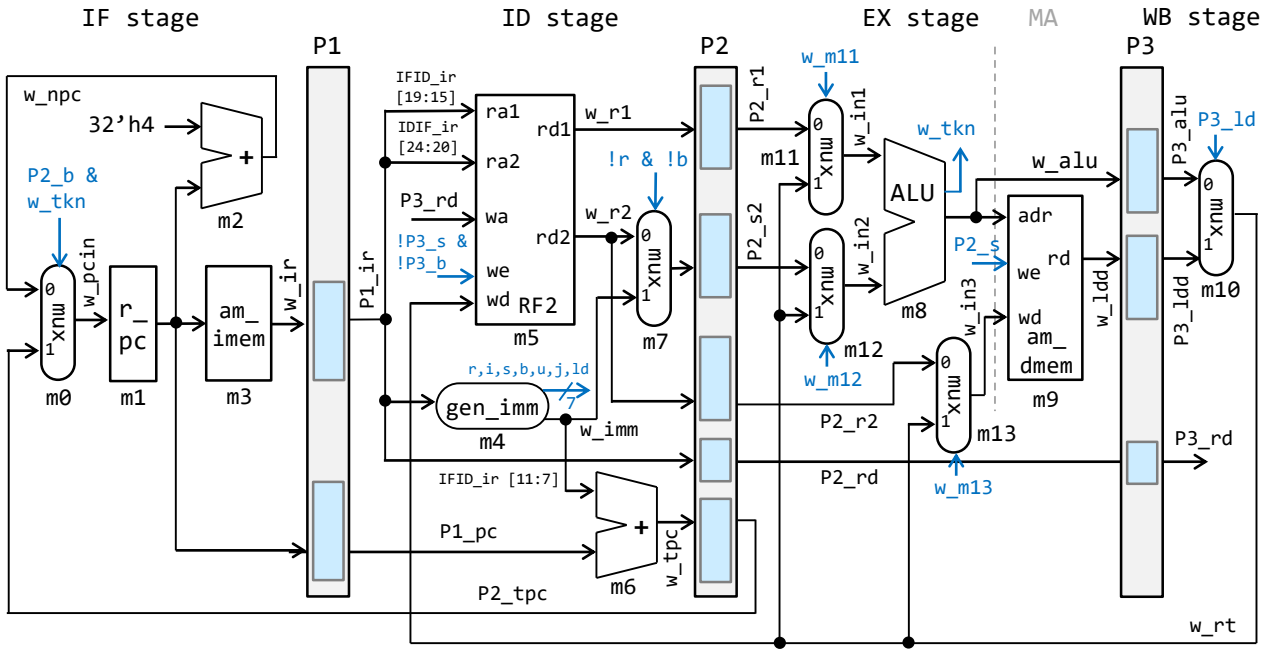



```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  reg P1_v=0, P2_v=0, P3_v=0;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin, w_in1, w_in2, w_in3;
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld, w_tkn;
  reg [31:0] r_pc = 0; // m1
  wire w_miss = P2_b & w_tkn & P2_v;
  assign w_pcin = (w_miss) ? P2_tpc : w_npc; // m0
  assign w_npc = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  m_gen_imm m4 (P1_ir, w_imm, w_r, w_i, w_s, w_b, w_u, w_j, w_ld);
  m_RF2 m5 (w_clk, P1_ir[19:15], P1_ir[24:20], w_r1, w_r2,
            P3_rd, !P3_s & !P3_b & P3_v, w_rt);
  assign w_tpc = P1_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  always @(posedge w_clk) begin
    {P1_v, P2_v, P3_v} <= {!w_miss, !w_miss & P1_v, P2_v};
    {r_pc, P1_ir, P1_pc, P2_pc} <= {w_pcin, w_ir, r_pc, P1_pc};
    {P2_r1, P2_r2, P2_s2, P2_tpc} <= {w_r1, w_r2, w_s2, w_tpc};
    {P2_r, P2_s, P2_b, P2_ld} <= {w_r, w_s, w_b, w_ld};
    {P2_rs2, P2_rs1, P2_rd} <= {P1_ir[24:15], P1_ir[11:7]};
    {P3_pc, P3_ld} <= {P2_pc, P2_ld};
    {P3_alu, P3_ldd, P3_rd} <= {w_alu, w_ldd, P2_rd};
  end
  assign w_alu = w_in1 + w_in2;   // m8
  assign w_tkn = w_in1 != w_in2;  // m8
  m_am_dmem m9 (w_clk, w_alu, P2_s & P2_v, w_in3, w_ldd);
  assign w_rt = (P3_ld) ? P3_ldd : P3_alu; // m10
  wire w_f =!P3_s & !P3_b & |P3_rd & P3_v;
  assign w_in1 = (w_f & P2_rs1==P3_rd) ? w_rt : P2_r1; // m11
  assign w_in3 = (w_f & P2_rs2==P3_rd) ? w_rt : P2_r2; // m13
  assign w_in2 = (w_f & P2_rs2==P3_rd & (P2_r|P2_b)) ? w_rt : P2_s2;// m12
endmodule
```



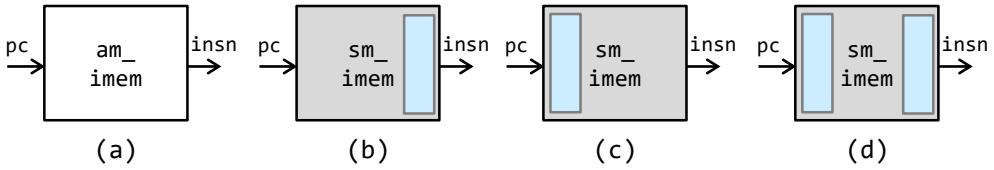

```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  reg P1_v=0, P2_v=0, P3_v=0;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin, w_in1, w_in2, w_in3;
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld, w_tkn;
  reg [31:0] r_pc = 0; // m1
  wire w_miss = P2_b & w_tkn & P2_v;
  assign w_pcin = (w_miss) ? P2_tpc : w_npc; // m0
  assign w_npc = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  m_gen_imm m4 (P1_ir, w_imm, w_r, w_i, w_s, w_b, w_u, w_j, w_ld);
  m_RF2 m5 (w_clk, P1_ir[19:15], P1_ir[24:20], w_r1, w_r2,
            P3_rd, !P3_s & !P3_b & P3_v, w_rt);
  assign w_tpc = P1_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  always @(posedge w_clk) begin
    {P1_v, P2_v, P3_v} <= {!w_miss, !w_miss & P1_v, P2_v};
    {r_pc, P1_ir, P1_pc, P2_pc} <= {w_pcin, w_ir, r_pc, P1_pc};
    {P2_r1, P2_r2, P2_s2, P2_tpc} <= {w_r1, w_r2, w_s2, w_tpc};
    {P2_r, P2_s, P2_b, P2_ld} <= {w_r, w_s, w_b, w_ld};
    {P2_rs2, P2_rs1, P2_rd} <= {P1_ir[24:15], P1_ir[11:7]};
    {P3_pc, P3_ld} <= {P2_pc, P2_ld};
    {P3_alu, P3_ldd, P3_rd} <= {w_alu, w_ldd, P2_rd};
  end
  assign w_alu = w_in1 + w_in2;   // m8
  assign w_tkn = w_in1 != w_in2;  // m8
  m_am_dmem m9 (w_clk, w_alu, P2_s & P2_v, w_in3, w_ldd);
  assign w_rt = (P3_ld) ? P3_ldd : P3_alu; // m10
  wire w_f =!P3_s & !P3_b & |P3_rd & P3_v;
  assign w_in1 = (w_f & P2_rs1==P3_rd) ? w_rt : P2_r1; // m11
  assign w_in3 = (w_f & P2_rs2==P3_rd) ? w_rt : P2_r2; // m13
  assign w_in2 = (w_f & P2_rs2==P3_rd & (P2_r|P2_b)) ? w_rt : P2_s2;// m12
endmodule
```

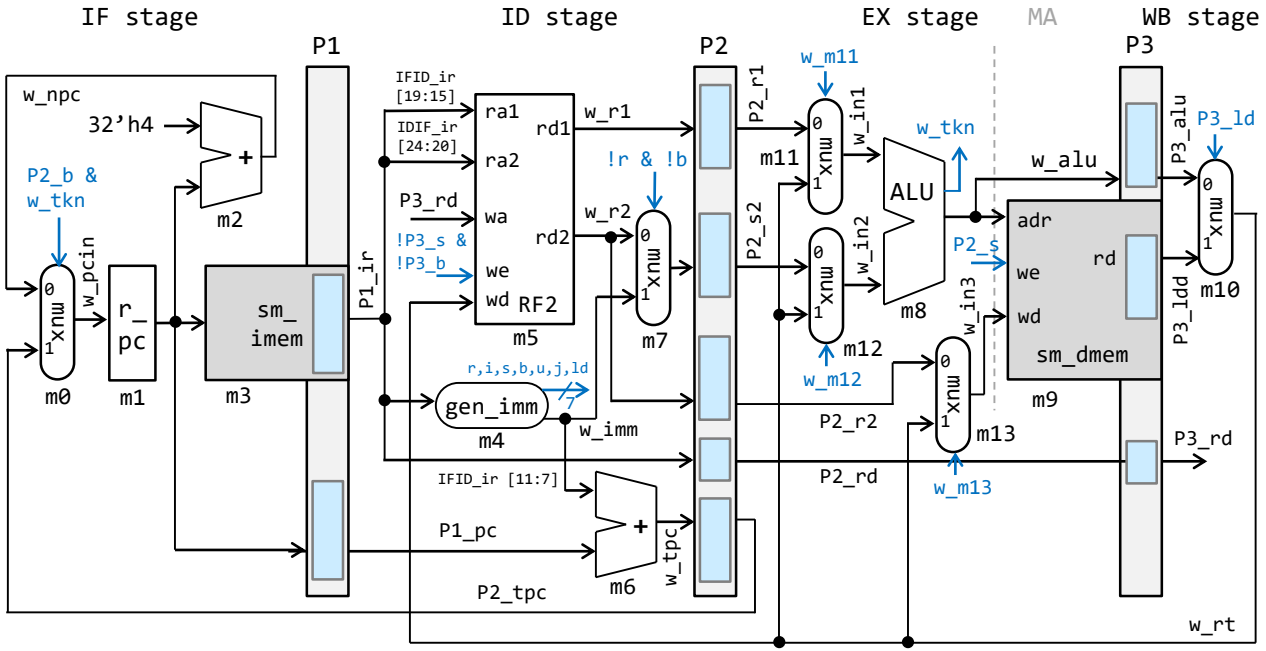
proc8: 4-stage pipelining processor



```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  reg P1_v=0, P2_v=0, P3_v=0;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin, w_in1, w_in2, w_in3;
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld, w_tkn;
  reg [31:0] r_pc = 0; // m1
  wire w_miss = P2_b & w_tkn & P2_v;
  assign w_pcin = (w_miss) ? P2_tpc : w_npc; // m0
  assign w_npc = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  m_gen_imm m4 (P1_ir, w_imm, w_r, w_i, w_s, w_b, w_u, w_j, w_ld);
  m_RF2 m5 (w_clk, P1_ir[19:15], P1_ir[24:20], w_r1, w_r2,
            P3_rd, !P3_s & !P3_b & P3_v, w_rt);
  assign w_tpc = P1_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  always @(posedge w_clk) begin
    {P1_v, P2_v, P3_v} <= {!w_miss, !w_miss & P1_v, P2_v};
    {r_pc, P1_ir, P1_pc, P2_pc} <= {w_pcin, w_ir, r_pc, P1_pc};
    {P2_r1, P2_r2, P2_s2, P2_tpc} <= {w_r1, w_r2, w_s2, w_tpc};
    {P2_r, P2_s, P2_b, P2_ld} <= {w_r, w_s, w_b, w_ld};
    {P2_rs2, P2_rs1, P2_rd} <= {P1_ir[24:15], P1_ir[11:7]};
    {P3_pc, P3_ld} <= {P2_pc, P2_ld};
    {P3_alu, P3_ldd, P3_rd} <= {w_alu, w_ldd, P2_rd};
  end
  assign w_alu = w_in1 + w_in2;   // m8
  assign w_tkn = w_in1 != w_in2;  // m8
  m_am_dmem m9 (w_clk, w_alu, P2_s & P2_v, w_in3, w_ldd);
  assign w_rt = (P3_ld) ? P3_ldd : P3_alu; // m10
  wire w_f =!P3_s & !P3_b & |P3_rd & P3_v;
  assign w_in1 = (w_f & P2_rs1==P3_rd) ? w_rt : P2_r1; // m11
  assign w_in3 = (w_f & P2_rs2==P3_rd) ? w_rt : P2_r2; // m13
  assign w_in2 = (w_f & P2_rs2==P3_rd & (P2_r|P2_b)) ? w_rt : P2_s2;// m12
endmodule
```

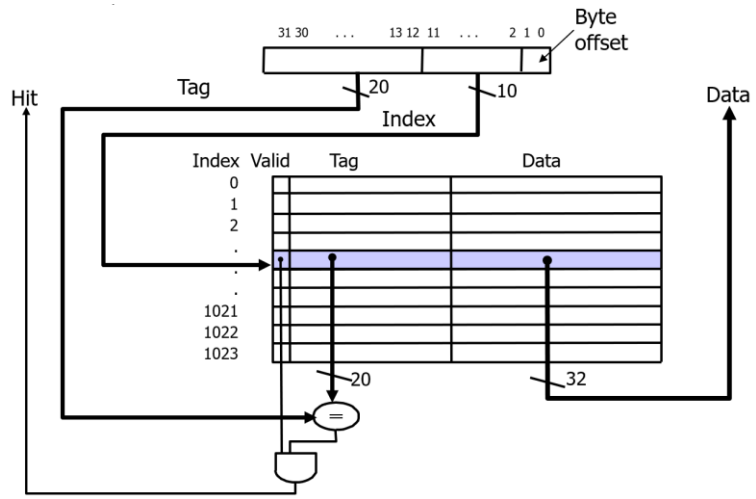
proc8: 4-stage pipelining processor





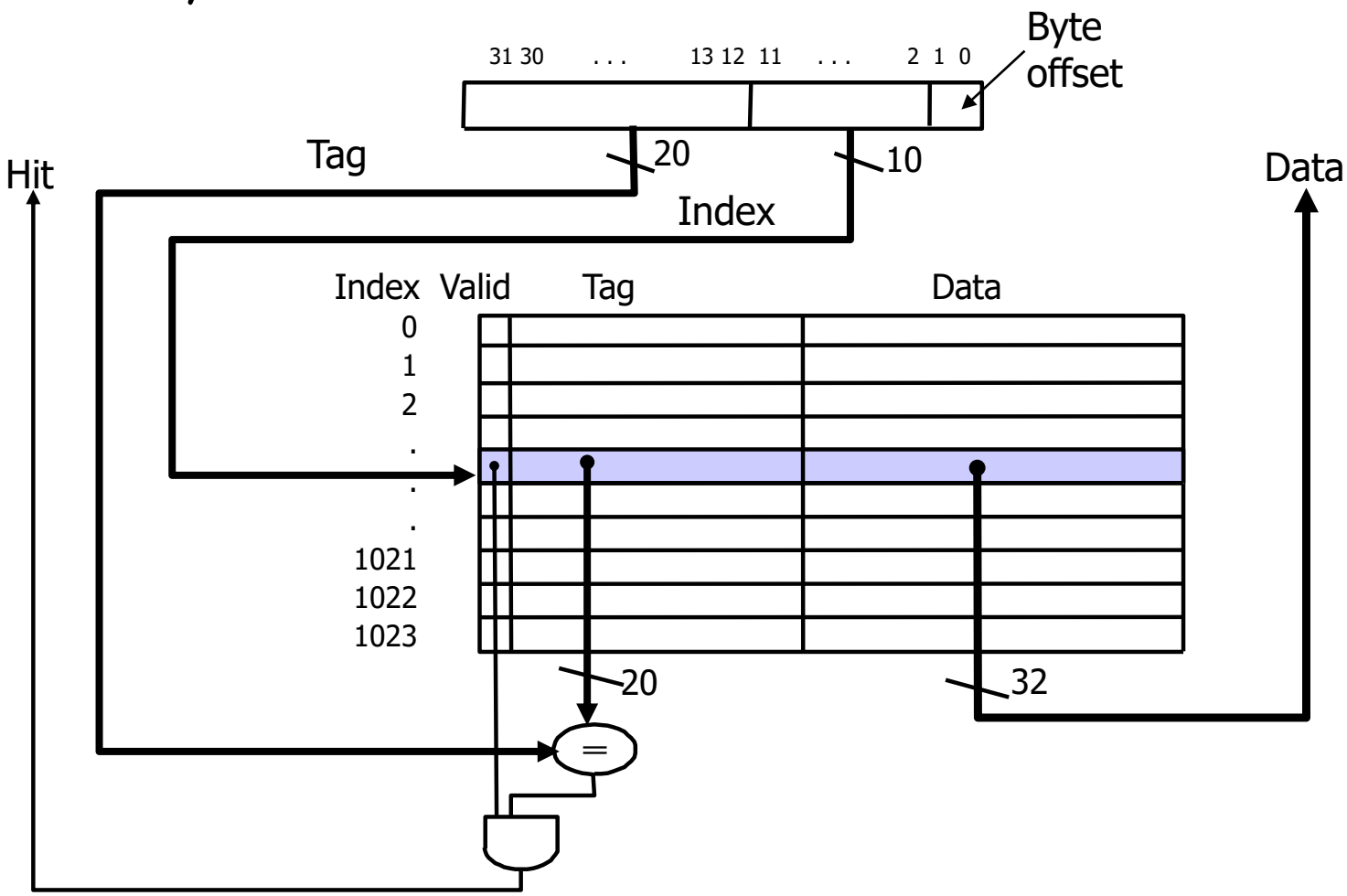
Cache1

```
module m_cache1(w_clk, w_adr, w_hit, w_dout, w_wadr, w_we, w_wd);
  input  wire w_clk, w_we;
  input  wire [31:0] w_adr;
  input  wire [4:0] w_wadr;
  input  wire [57:0] w_wd;
  output wire w_hit;
  output wire [31:0] w_dout;
  wire w_v;
  wire [24:0] w_tag;
  reg [57:0] mem [0:31];
  always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
  assign {w_v, w_tag, w_dout} = mem[w_adr[6:2]];
  assign w_hit = w_v & (w_tag==w_adr[31:7]);
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;
endmodule
```



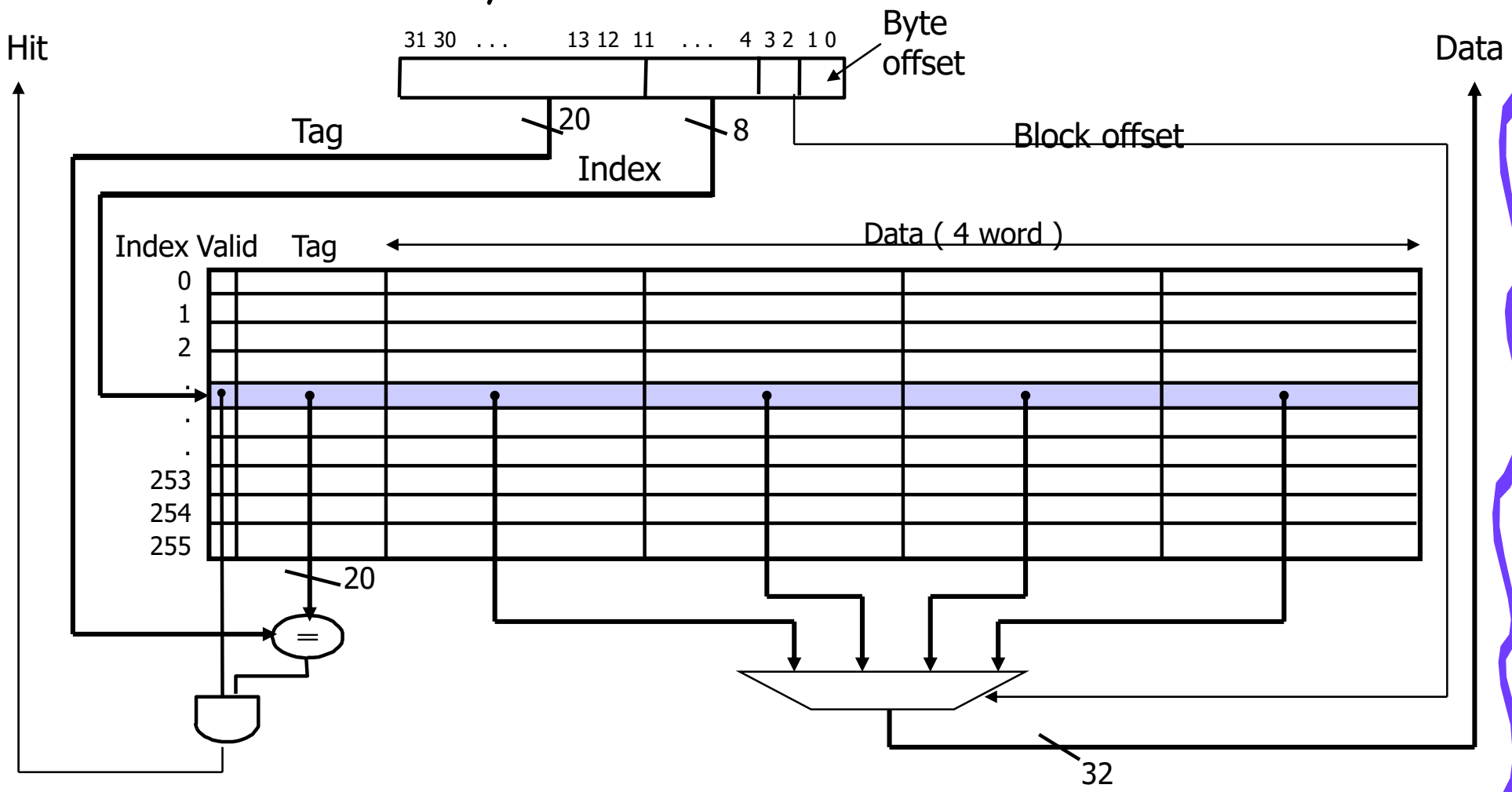
XXX Cache

- One word/block, cache size = 1K words



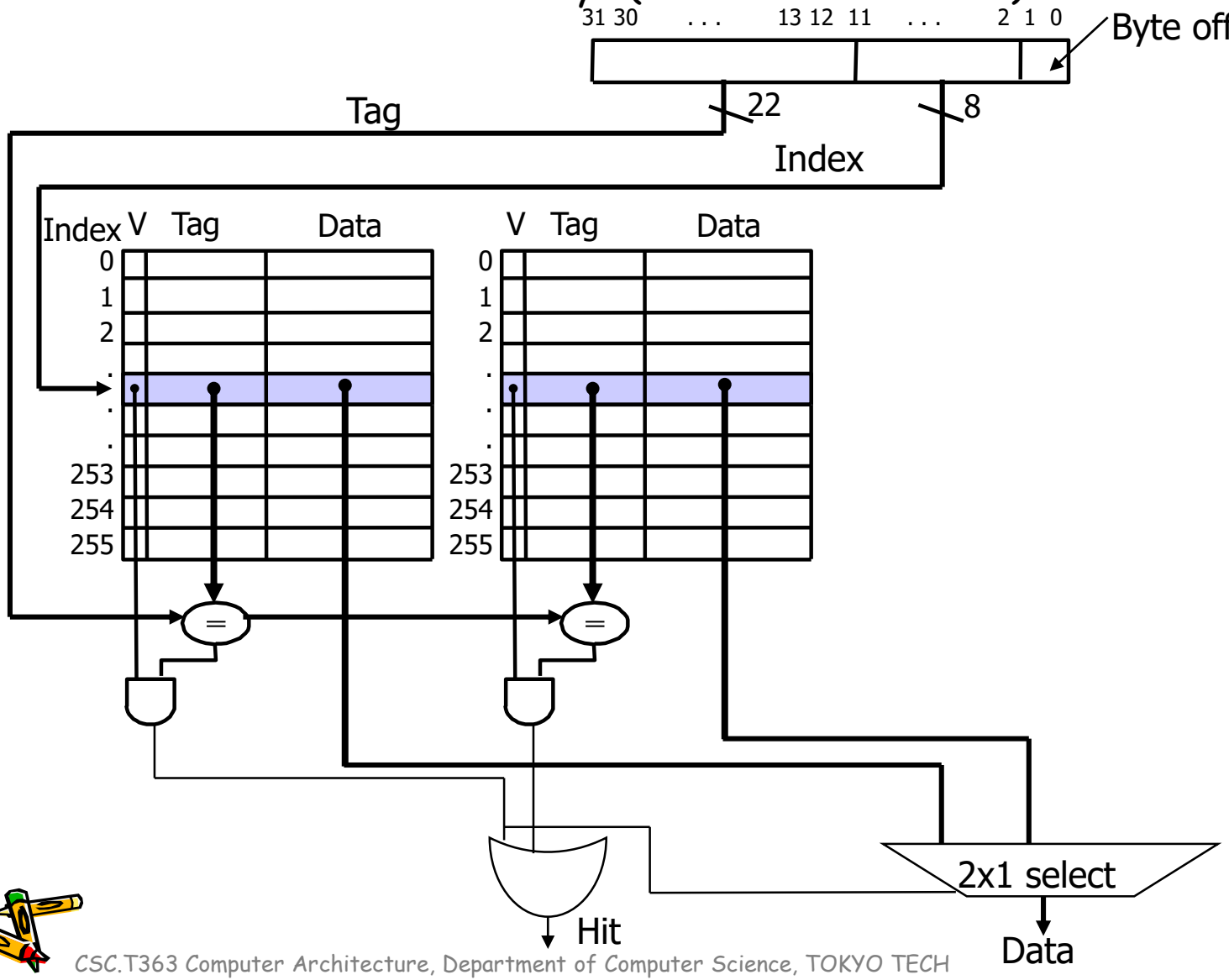
XXX Cache

- Four words/block, cache size = 1K words



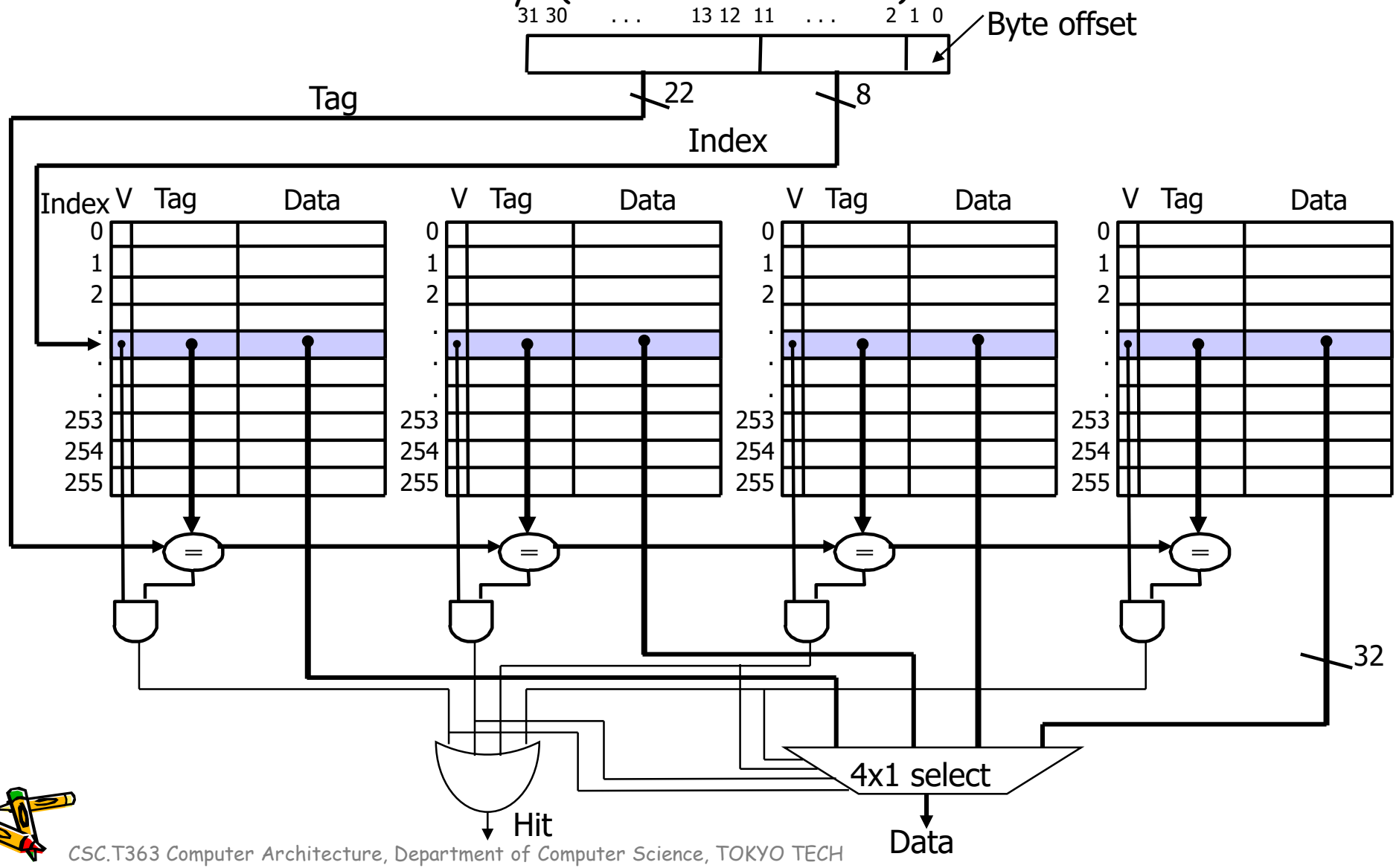
XXX Cache

2^8 = **256 sets** each with XX ways (each with one block)

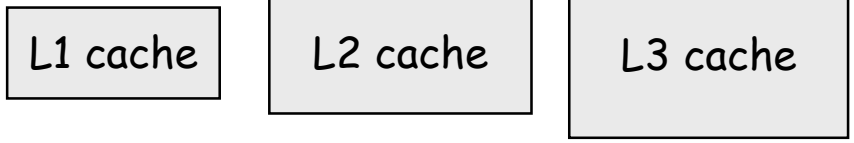


XXX Cache

2^8 = **256 sets** each with XX ways (each with one block)



Reducing Cache Miss Rates by **multiple levels**



Enough room on the die for **bigger L1 caches** *or* for a **second level of caches** – normally a **unified** L2 cache (i.e., it holds both instructions and data) and in some cases even a **unified L3 cache**

For our example,

CPI_{ideal} of 2,

100 cycle miss penalty (to main memory),

36% load/stores,

a 2% (4%) L1I$ (D$) miss rate,

add a UL2$ that has a 25 cycle miss penalty and a 0.5% miss rate

$$CPI_{stalls} = 2 + .02 \times \mathbf{25} + .36 \times .04 \times \mathbf{25} + \mathbf{.005 \times 100} + \mathbf{.36 \times .005 \times 100} = 3.54$$

(as compared to **5.44** with no L2$)

Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are very different
  - Primary cache should focus on **minimizing hit time** in support of a shorter clock cycle
  - Secondary cache should focus on **reducing miss rate** to reduce the penalty of long main memory access times

- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate
- For the L2 cache, hit time is less important than miss rate
  - The L2$ hit time determines L1$'s miss penalty

Key Cache Design Parameters

	L1 typical	L2 typical
Total size (blocks)	250 to 2000	4000 to 250,000
Total size (KB)	16 to 64	500 to 8000
Block size (B)	32 to 64	32 to 128
Miss penalty (clocks)	10 to 25	100 to 1000
Miss rates	2% to 5%	0.1% to 2%

Two Machines' Cache Parameters

	Intel P4	AMD Opteron
L1 organization	Split I$ and D$	Split I$ and D$
L1 cache size	8KB for D$, 96KB for trace cache (~I$)	64KB for each of I$ and D$
L1 block size	64 bytes	64 bytes
L1 associativity	4-way set assoc.	2-way set assoc.
L1 replacement	~ LRU	LRU
L1 write policy	write-through	write-back
L2 organization	Unified	Unified
L2 cache size	512KB	1024KB (1MB)
L2 block size	128 bytes	64 bytes
L2 associativity	8-way set assoc.	16-way set assoc.
L2 replacement	~LRU	~LRU
L2 write policy	write-back	write-back

The Cache Design Space

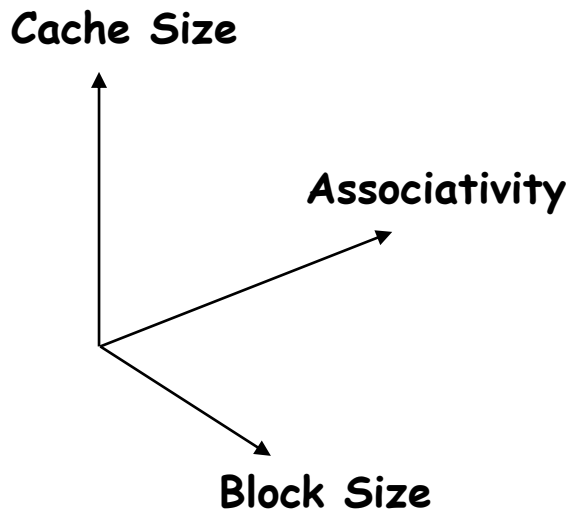
Several interacting dimensions

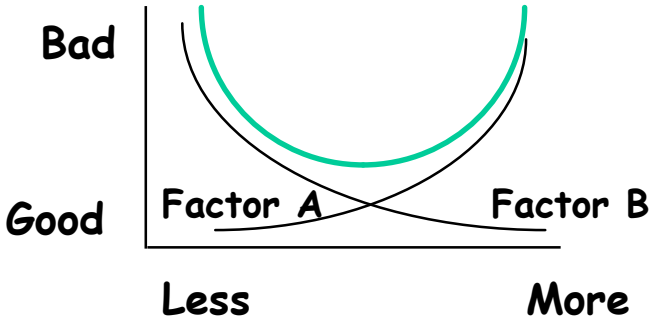
- cache size
- block size
- associativity
- replacement policy
- write-through vs write-back
- write allocation

The optimal choice is a compromise

- depends on access characteristics
  - workload
  - I-cache, D-cache
- depends on technology / cost

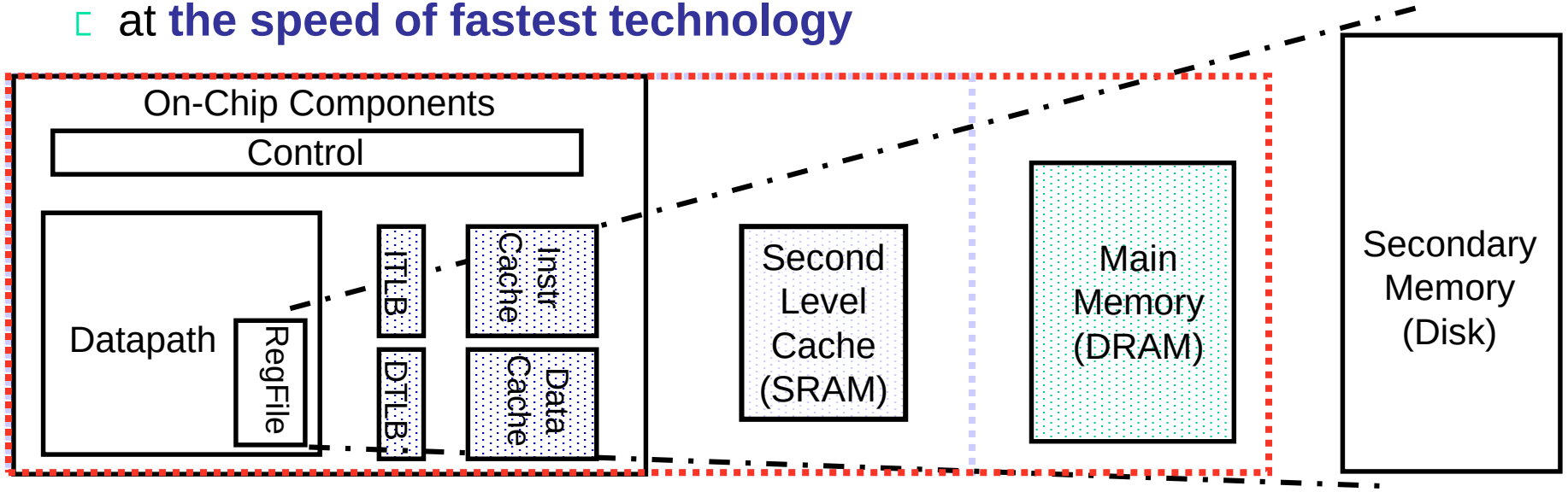
Simplicity often wins





A Typical Memory Hierarchy

- By taking advantage of **the principle of locality**（局所性）
  - Present **much memory** in **the cheapest technology**
  - at **the speed of fastest technology**



Speed (%cycles):	½'s	1's	10's	100's	1,000's
Size (bytes):	100's	K's	10K's	M's	G's to T's
Cost:	highest				lowest

TLB: Translation Lookaside Buffer

DDR4 SDRAM

- 規格：DDR4 デスクトップ用 動作電圧：1.2v JEDEC準拠品（XMP2.0非搭載）
- 速度：PC4-25600 3200Mhz CL値：22-22-22-52 / 容量：32GBx2枚（64G
- 対応チップセット：Intel：Z590/H570/B560/H510/Z490 ・AMD：

チップ規格	モジュール規格	メモリクロック (MHz)	バスクロック (MHz)	転送速度 (GB/秒)	JEDEC 規格
DDR4-800	PC4-6400	50	400	6.4	
DDR4-1066	PC4-8528	66	533	8.5	
DDR4-1333	PC4-10664	83	666	10.6	
DDR4-1600	PC4-12800	100	800	12.8	○
DDR4-1866	PC4-14900	116	933	14.9	○
DDR4-2133	PC4-17000	133	1066	17.0	○
DDR4-2400	PC4-19200	150	1200	19.2	○
DDR4-2666	PC4-21333	166	1333	21.3	○
DDR4-2800	PC4-22400	175	1400	22.4	
DDR4-2933	PC4-23466	183	1466	23.4	○
DDR4-3000	PC4-24000	188	1500	24.0	
DDR4-3200	PC4-25600	200	1600	25.6	○
DDR4-3300	PC4-26400	206	1650	26.4	

Amazon, Wikipedia

DDR SDRAM Latency

テクノロジー	モジュール速度 (MT/s)	クロックサイクル時間（ns）	CASレイテンシー	レイテンシー（ns）
SDR	100	8.00	3	24.00
SDR	133	7.50	3	22.50
DDR	333	6.00	2.5	15.00
DDR	400	5.00	3	15.00
DDR2	667	3.00	5	15.00
DDR2	800	2.50	6	15.00
DDR3	1333	1.50	9	13.50
DDR3	1600	1.25	11	13.75
DDR4	1866	1.07	13	13.93
DDR4	2133	0.94	15	14.06
DDR4	2400	0.83	17	14.17
DDR4	2666	0.75	19	14.25
DDR4	2933	0.68	21	14.32
DDR4	3200	0.62	22	13.75
DDR5	4800	0.42	40	16.67

https://www.crucial.jp/articles/about-memory/difference-between-speed-and-latency

Latest Academic Paper

Register File Prefetching

Sudhanshu Shukla
sudhanshu.shukla@intel.com
Processor Architecture Research Lab, Intel Labs

Sumeet Bandishte
sumeet.bandishte@intel.com
Processor Architecture Research Lab, Intel Labs

Jayesh Gaur
jayesh.gaur@intel.com
Processor Architecture Research Lab, Intel Labs

Sreenivas Subramoney
sreenivas.subramoney@intel.com
Processor Architecture Research Lab, Intel Labs

ISCA '22, June 18–22, 2022, New York, NY, USA

4 EVALUATION METHODOLOGY

We simulate a dynamically scheduled x86 core using an in-house, execution driven cycle-accurate simulator that models a 5-wide OOO core clocked at 4 GHz. The core parameters are similar to the

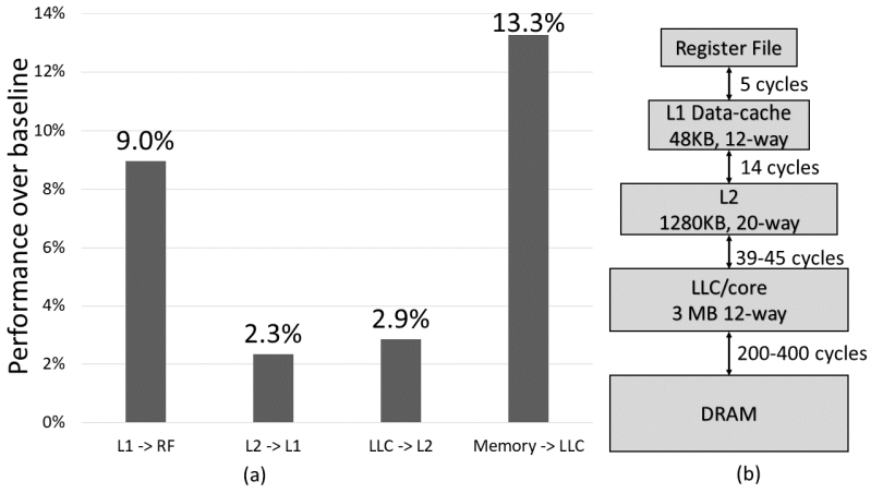
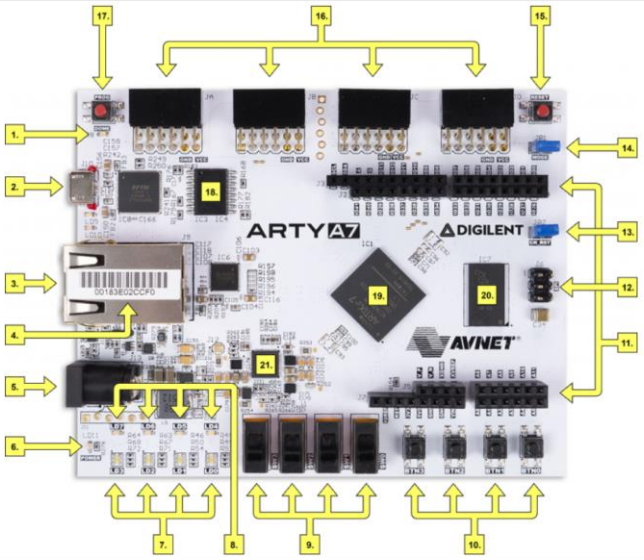


Figure 1: Performance headroom from oracle prefetching across the different levels of cache and memory hierarchy for a processor similar to Intel Tiger Lake [22]. An oracle prefetching from level N to level N-1 will ensure all hits at level N will be served at the latency of level N -1. Prefetching between L1 to RF and between Memory to LLC provide the highest headroom.

Arty A7 and server computer of ACRi Room



Callout	Description	Callout	Description	Callout	Description
1	FPGA programming DONE LED	8	User RGB LEDs	15	chipKIT processor reset
2	Shared USB JTAG / UART port	9	User slide switches	16	Pmod connectors
3	Ethernet connector	10	User push buttons	17	FPGA programming reset button
4	MAC address sticker	11	Arduino/chipKIT shield connectors	18	SPI flash memory
5	Power jack for optional external supply	12	Arduino/chipKIT shield SPI connector	19	Artix FPGA
6	Power good LED	13	chipKIT processor reset jumper	20	Micron DDR3 memory
7	User LEDs	14	FPGA programming mode	21	Dialog Semiconductor DA9062 power supply

FPGA Server

- CPU: Core i9 (**8 core /16 thread**)
- Memory: DDR4 **128GB** (32GB x 4)
- Storage: SSD M.2 **1TB x 2**

Arty A7-35T (XC7A35TICSG324-1L, Artix-7 device)

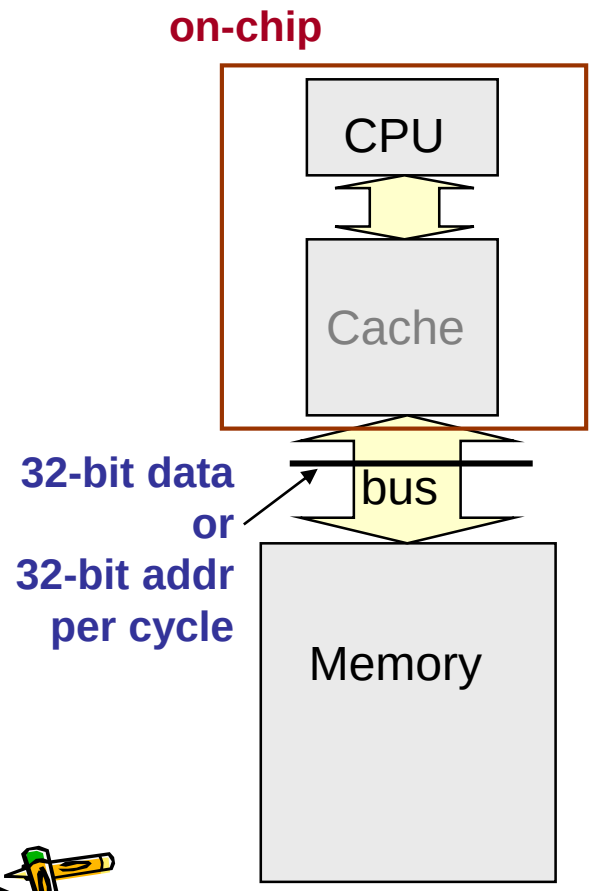
Arty A7: Artix-7 FPGA Development Board

$159.00

★★★★☆ (22 reviews) Write a Review

Memory Systems that Support Caches

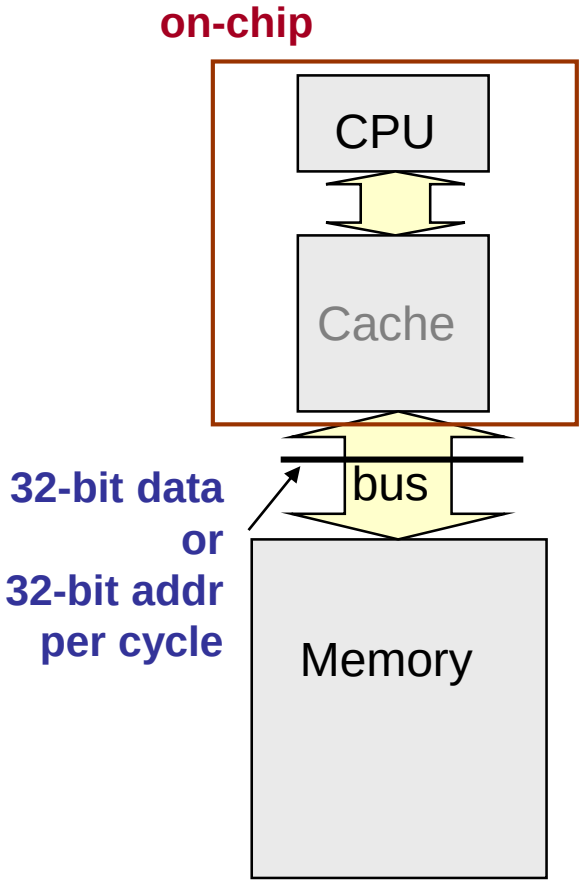
- The off-chip interconnect and memory architecture can affect overall system performance **in dramatic ways**



One word wide organization (one word wide bus and one word wide memory)

- **Assume**
  1. **1** clock cycle to send the address
  2. **25** clock cycles for DRAM **cycle time**, **8** clock cycles **access time**
  3. **1** clock cycle to return a word of data
- **Memory-Bus** to Cache **bandwidth**
  - **number of bytes transferred from memory to cache per clock cycle**

One Word Wide Memory Organization

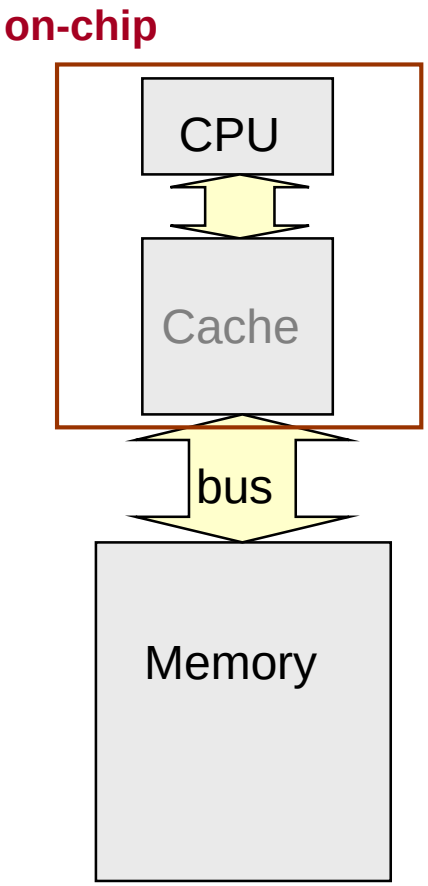


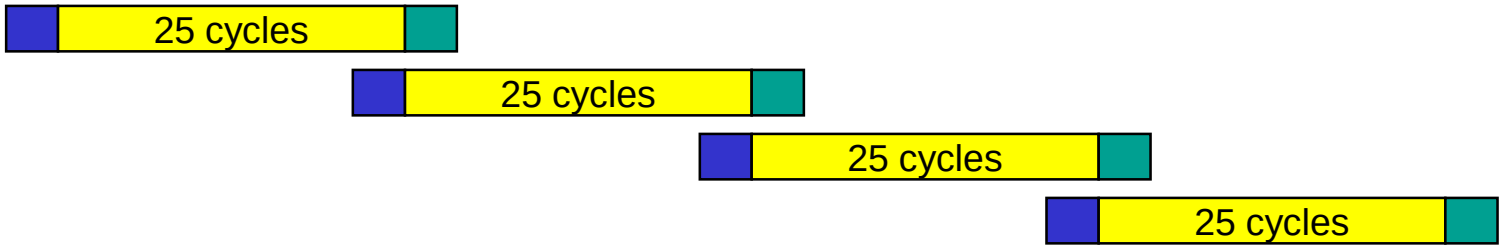
- The pipeline stalls the number of cycles for **one word** (32bit) from memory
  - **1** cycle to send address
  - **25** cycles to read DRAM
  - **1** cycle to return data
  - **27** **total clock cycles** miss penalty

- Number of bytes transferred per clock cycle (**bandwidth**) for a single miss
  - **4 / 27 = 0.148** bytes per clock

One Word Wide Memory Organization, con't

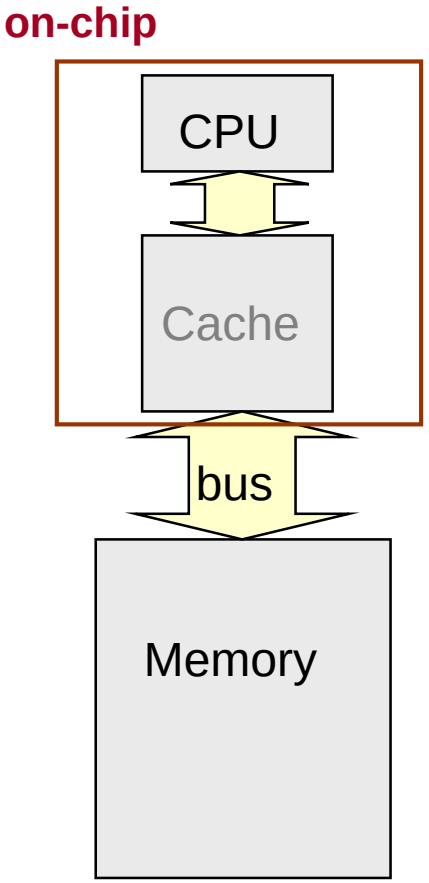
- What if the block size is **four words**?
  - **1** cycle to send 1st address
  - **4 * 25 = 100** cycles to read DRAM
  - **1** cycle to return last data word
  - **102 total clock cycles** miss penalty



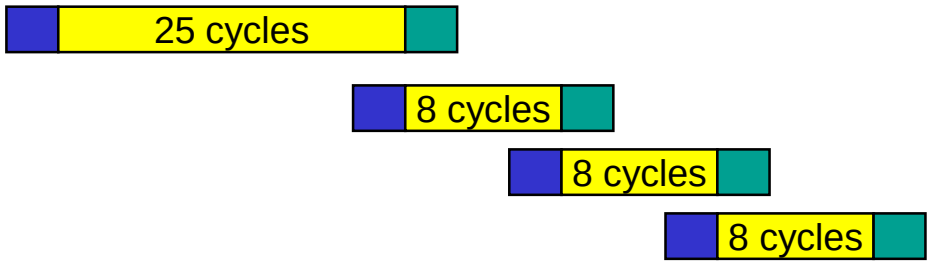


- Number of bytes transferred per clock cycle (**bandwidth**) for a single miss
  - **(4 x 4) / 102 = 0.157** bytes per clock

One Word Wide Memory Organization, con't

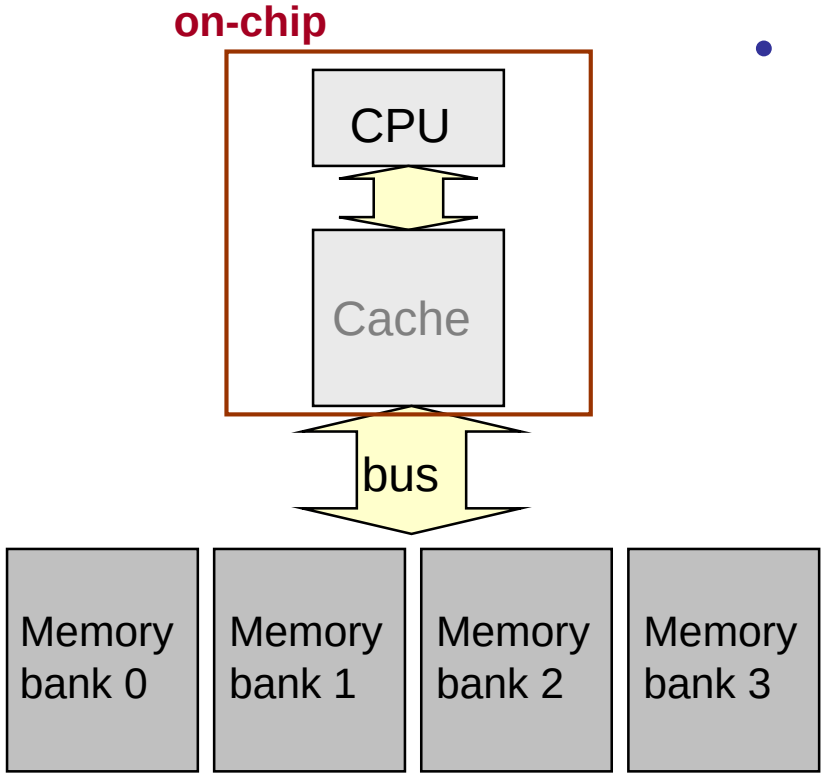


- What if the block size is **four words** and if a **page mode DRAM** is used?
  - **1** cycle to send 1st address
  - **25 + (3 * 8) = 49** cycles to read DRAM
  - **1** cycle to return last data word
  - **51 total clock cycles** miss penalty



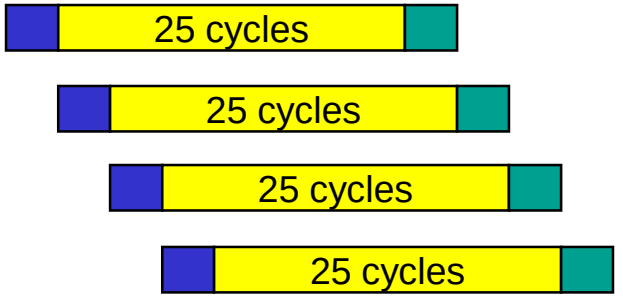
- Number of bytes transferred per clock cycle (**bandwidth**) for a single miss
  - **(4 x 4) / 51 = 0.314** bytes per clock

Interleaved（インターリーブ）Memory Organization



With **parallelism**

- For a block size of **four words** with **interleaved memory (4 banks)**
  - **1** cycle to send 1st address
  - **25 + 3 = 28** cycles to read DRAM
  - **1** cycle to return last data word
  - **30 total clock cycles** miss penalty



- Number of bytes transferred per clock cycle (bandwidth) for a single miss
  - **(4 x 4) / 30 = 0.533 bytes per clock**