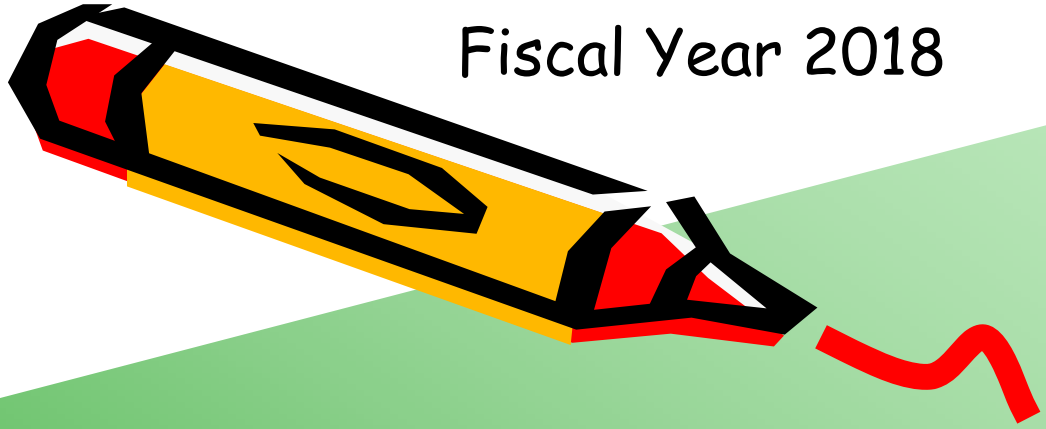


Fiscal Year 2018

Ver. 2018-12-16a

Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

5. Instruction Level Parallelism: Concepts and Challenges

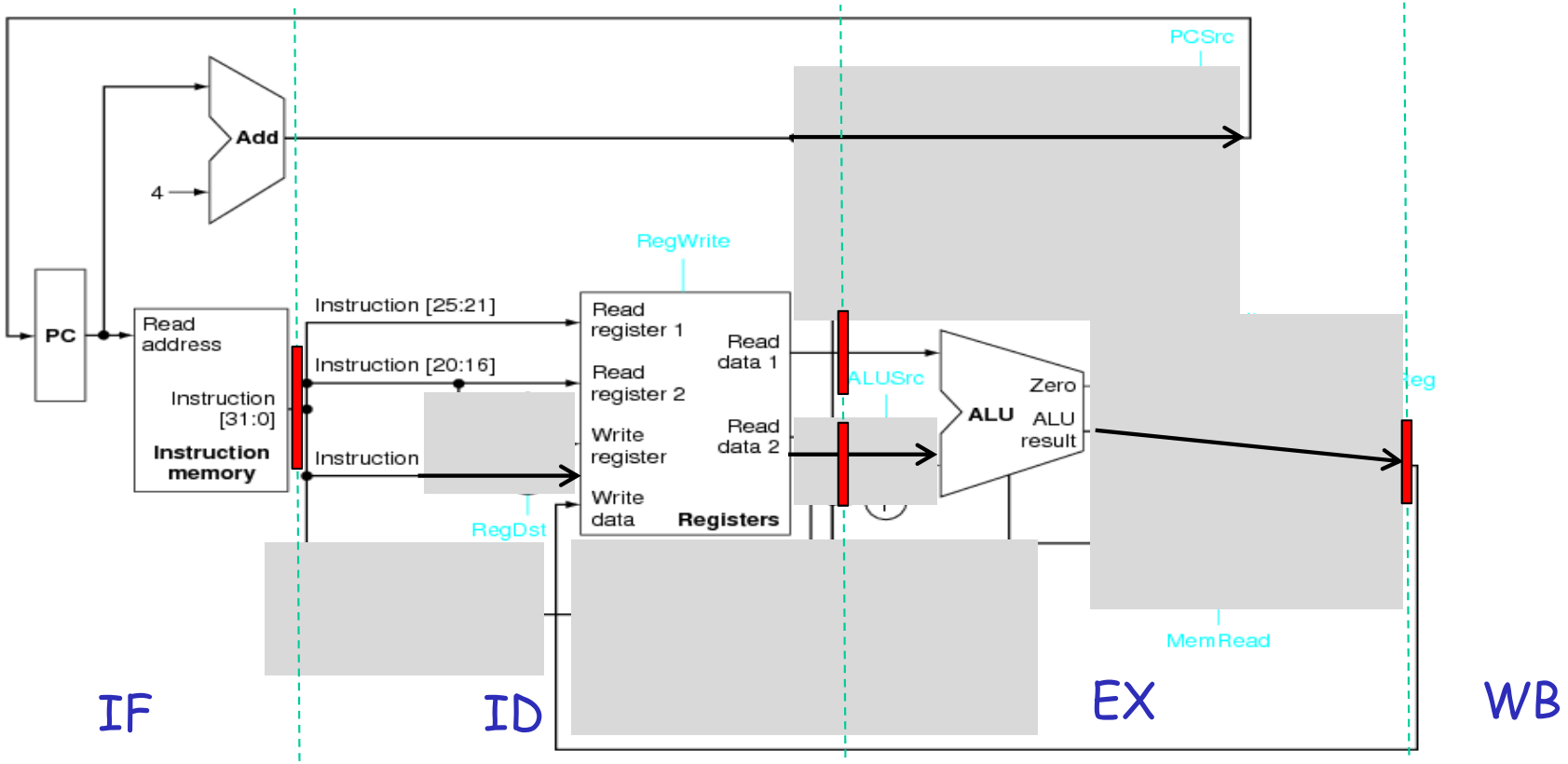
www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 13:20-14:50, Thr 13:20-14:50

Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

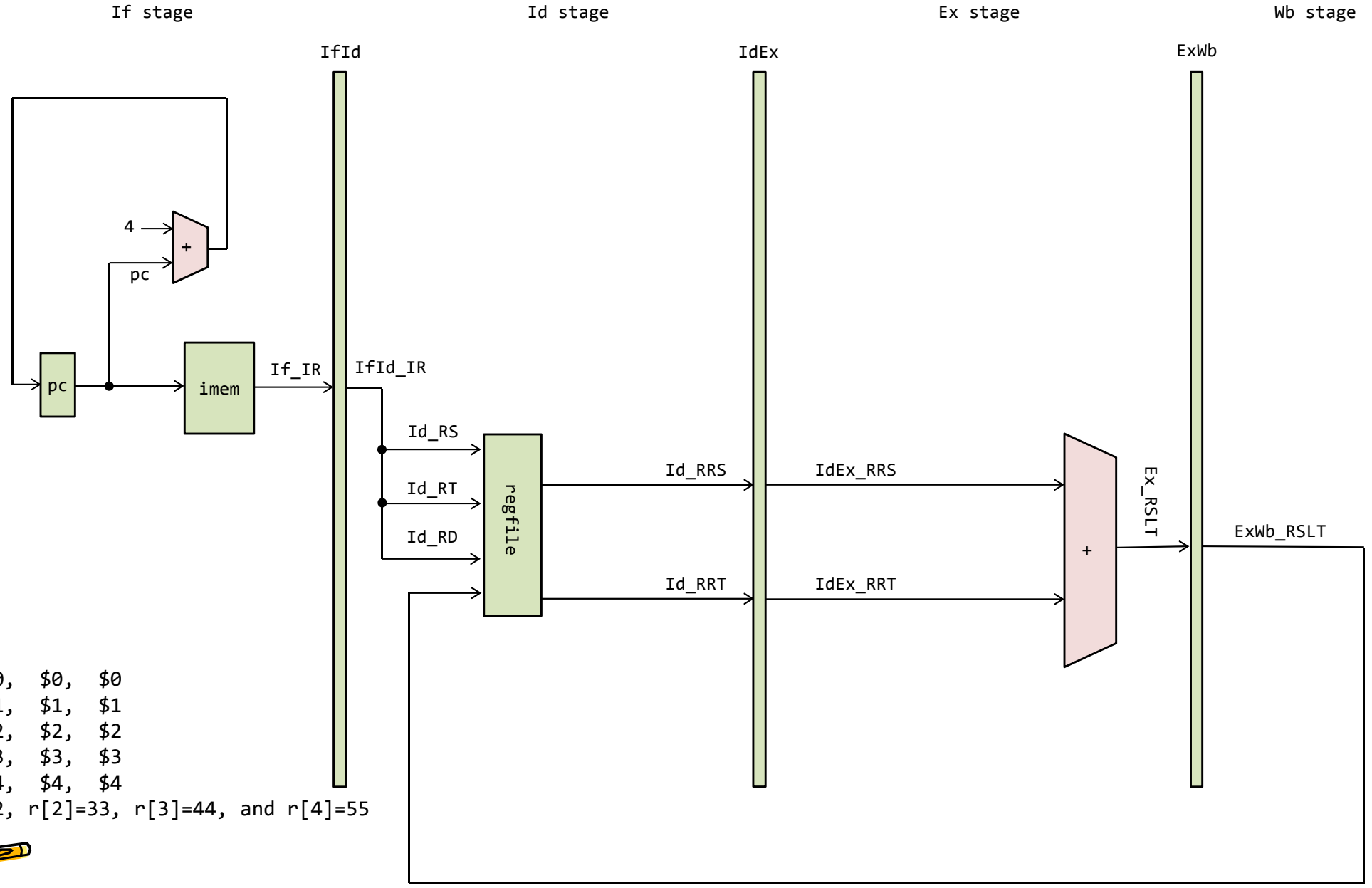
Pipeline registers

- `add $0, $0, $0 # NOP, $0 <= 0 + 0`
- `add $1, $1, $1 # $1 <= 22 + 22`
- `add $2, $2, $2 # $2 <= 33 + 33`
- `add $0, $0, $0 # NOP`
- `add $0, $0, $0 # NOP`
- `add $0, $0, $0 # NOP`

assuming initial values of r[1]=22 and r[2]=33



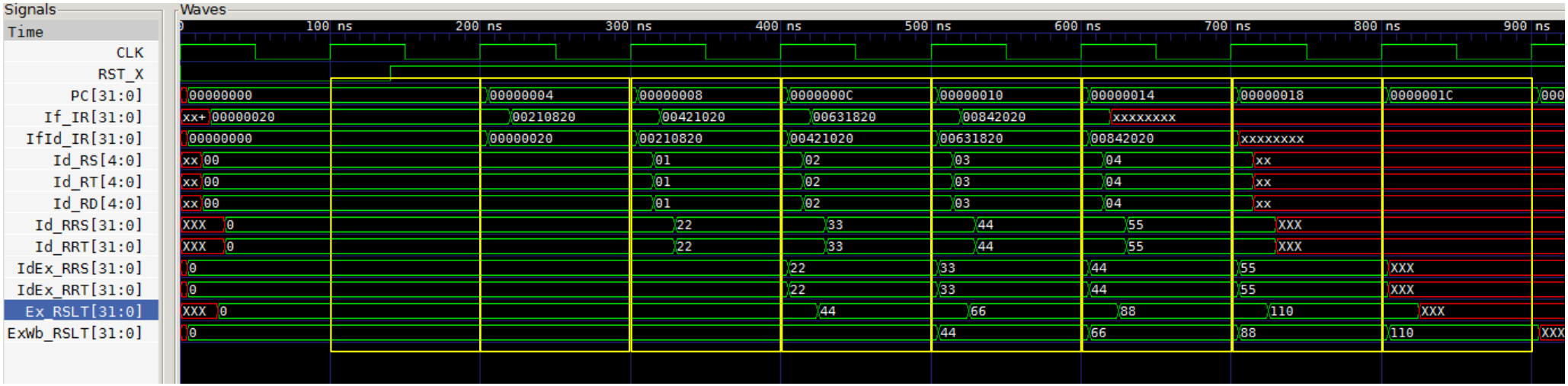
Four stage pipelined processor supporting ADD, which uses wrong RD and does not adopt data forwarding (proc05.v)




```
add  $0,  $0,  $0
add  $1,  $1,  $1
add  $2,  $2,  $2
add  $3,  $3,  $3
add  $4,  $4,  $4
r[1]=22, r[2]=33, r[3]=44, and r[4]=55
```

Waveform of Proc05

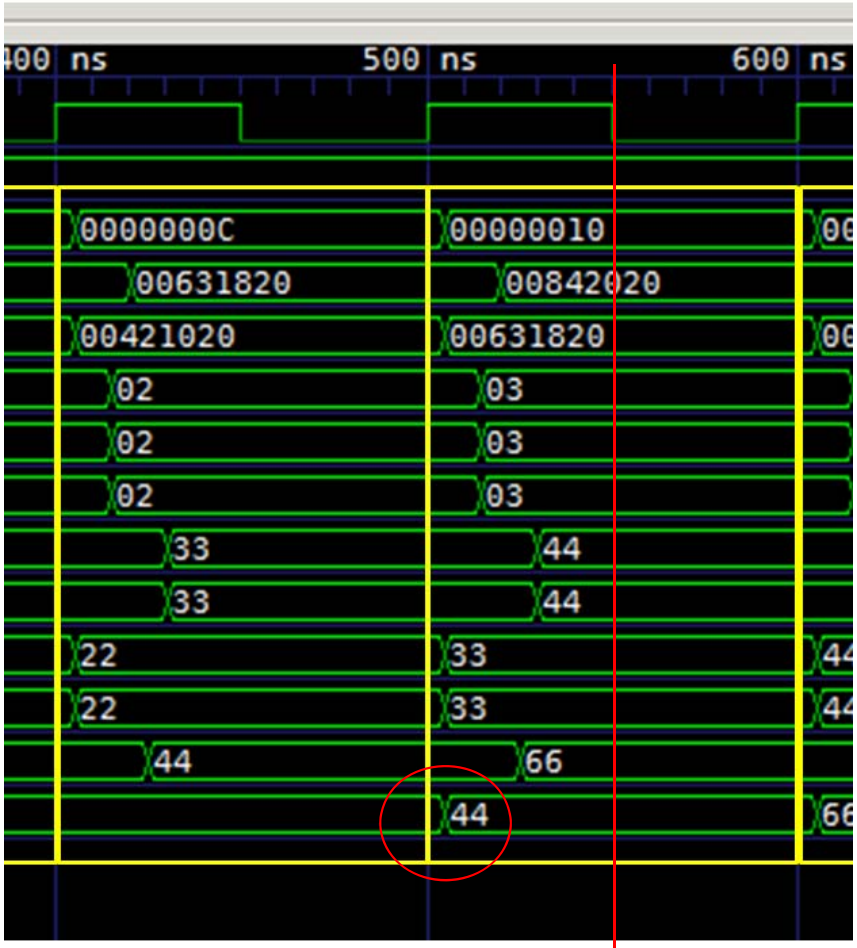
- Please confirm that the values in regfile are wrong




```
0x00 add  $0,  $0,  $0     IF        ID        EX        WB
0x04 add  $1,  $1,  $1               IF        ID        EX        WB
0x08 add  $2,  $2,  $2                         IF        ID        EX        WB
0x0C add  $3,  $3,  $3                                   IF        ID        EX        WB
0x10 add  $4,  $4,  $4                                             IF        ID        EX        WB
```

r[1]=22, r[2]=33, r[3]=44, and r[4]=55

Waveform of Proc05 towards Proc06




```
/* 32bitx32 2R/1W General Purpose Registers (Register File)
/*****************************************************************************
module GPR(CLK, REGNUM0, REGNUM1, REGNUM2, DIN0, WE0, DOUT0, DOUT1);
  input  wire        CLK;
  input  wire  [4:0] REGNUM0, REGNUM1, REGNUM2;
  input  wire [31:0] DIN0;
  input  wire        WE0;
  output wire [31:0] DOUT0, DOUT1;

  reg [31:0] r[0:31];
  assign #15 DOUT0 = (REGNUM0==0) ? 0 : r[REGNUM0];
  assign #15 DOUT1 = (REGNUM1==0) ? 0 : r[REGNUM1];
  always @(negedge CLK) if(WE0) r[REGNUM2] <= #10 DIN0;
//  always @(posedge CLK) if(WE0) r[REGNUM2] <= #10 DIN0;
endmodule
/*****************************************************************************
```

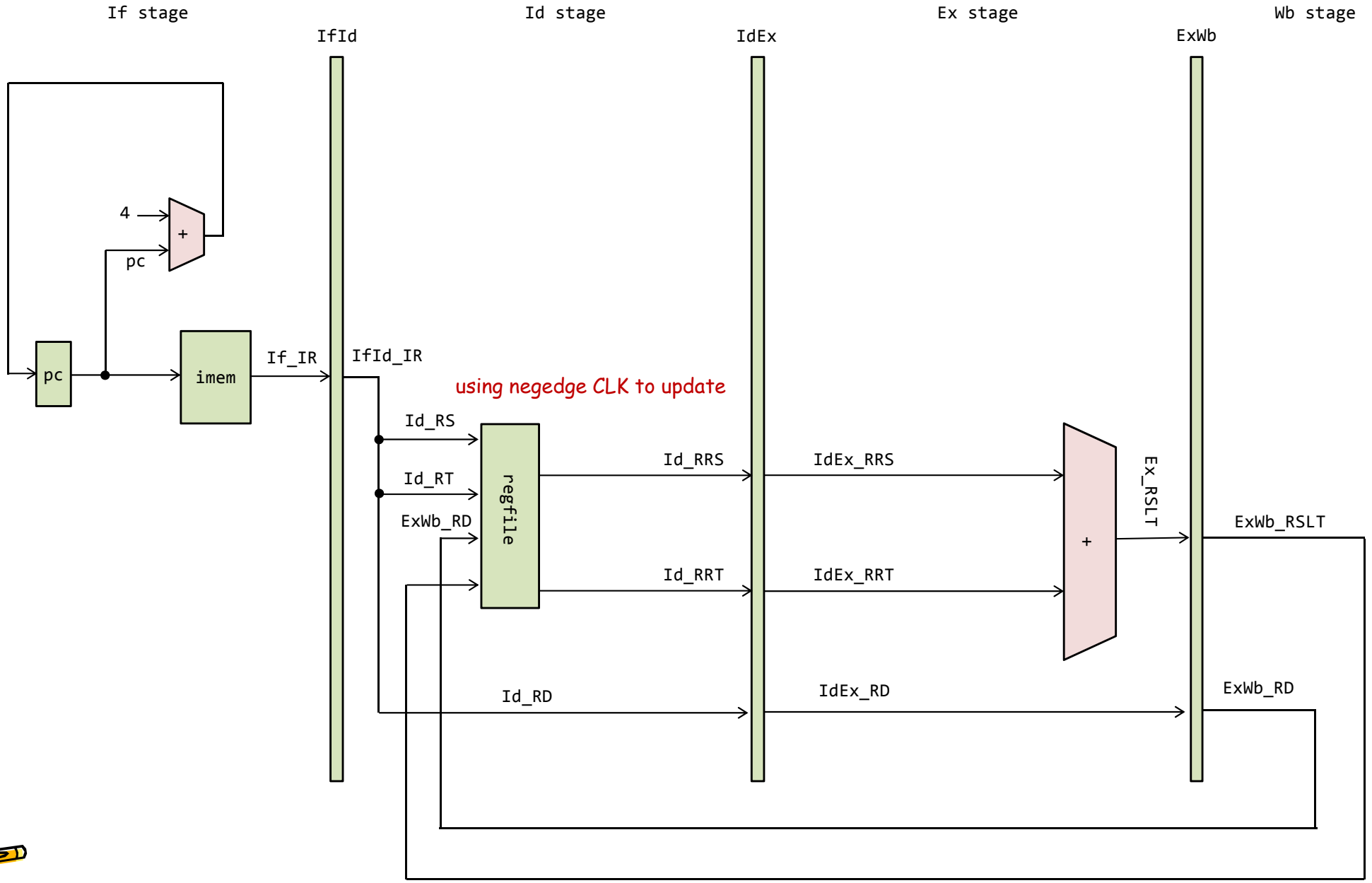
using negedge CLK to update regfile

WB

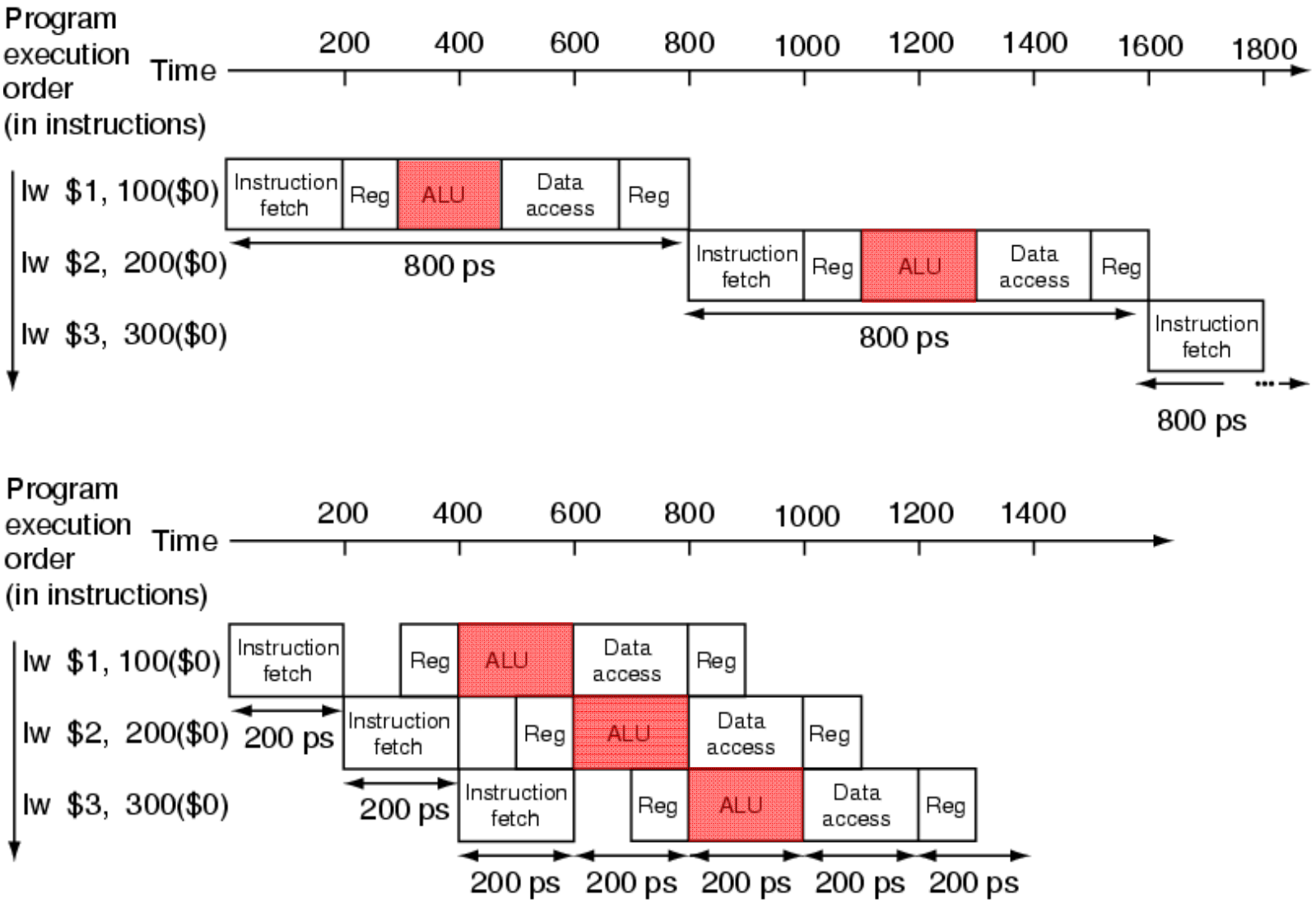
EX WB

ID EX

Four stage pipelined processor supporting ADD, which does not adopt data forwarding (proc06.v, Homework 4)

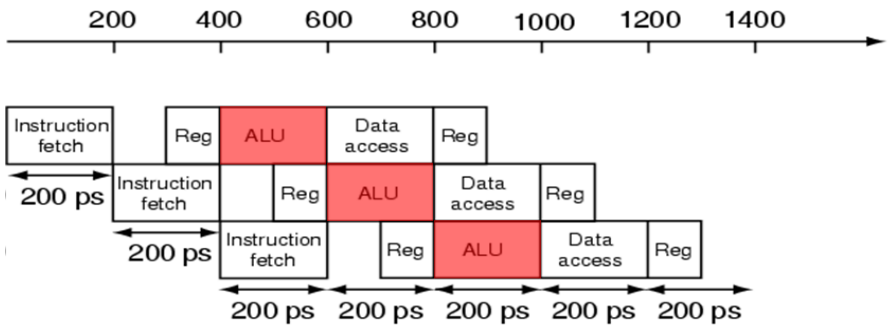


Single-cycle and pipelined processors

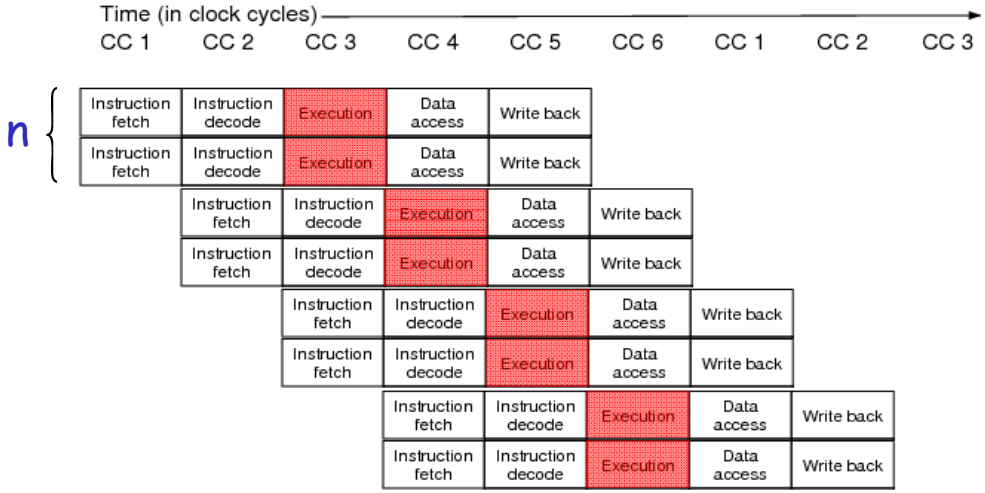


Scalar and Superscalar processors

- Scalar processor can execute at most one single instruction per clock cycle using one ALU.
  - IPC (Executed Instructions Per Cycle) is less than 1.
- Superscalar processor can execute more than one instruction per clock cycle by executing multiple instructions using multiple pipelines.
  - IPC (Executed Instructions Per Cycle) can be more than 1.
  - using n pipelines is called n-way superscalar



(a) pipeline diagram of scalar processor



(b) pipeline diagram of 2-way superscalar processor

Exercise: datapath of a 2-way superscalar

- Datapath of a 2-way superscalar processor supporting ADD, which does not adopt data forwarding

Homework 5

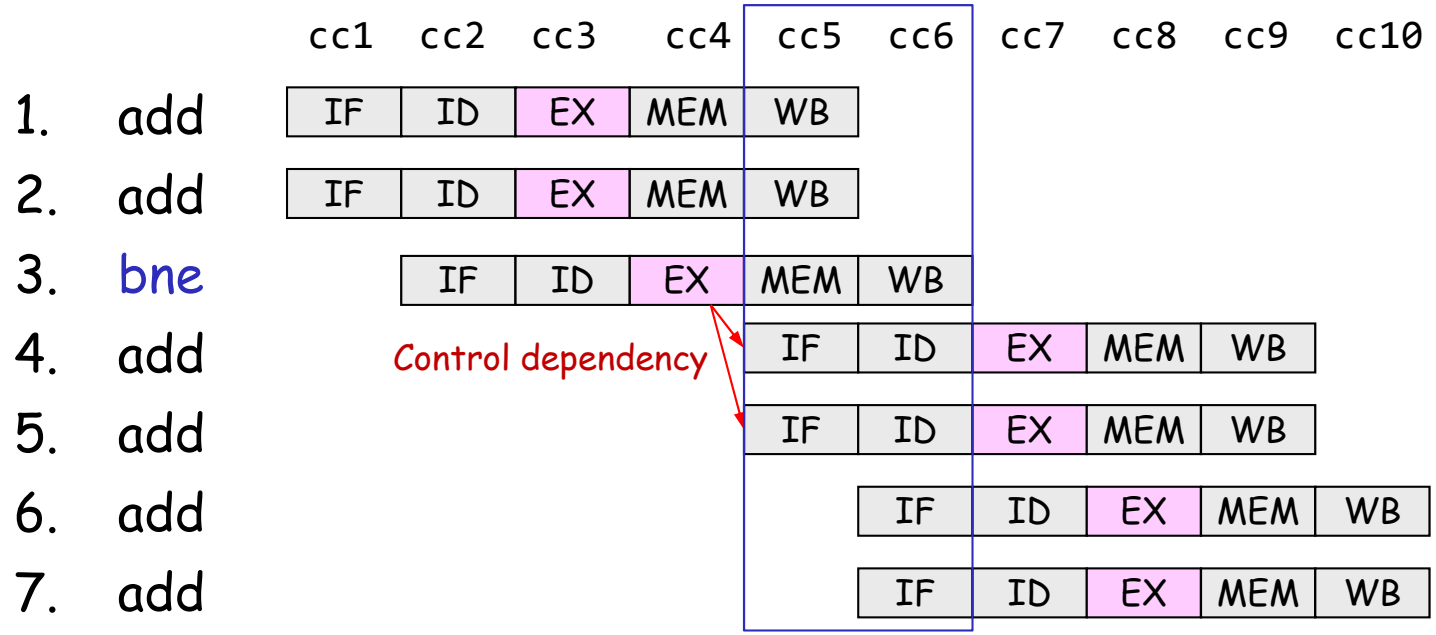
1. Design a four stage pipelined 2-way superscalar processor supporting MIPS add instruction in Verilog HDL. Please download proc06.v from the support page and refer it.
2. Verify the behavior of designed processor using following assembly code
   assuming initial values of r[1]=22, r[2]=33, r[3]=44, and r[4]=55
   - `add $0, $0, $0 #`
   - `add $0, $0, $0 #`
   - `add $1, $1, $1 #`
   - `add $2, $2, $2 #`
   - `add $3, $3, $3 #`
   - `add $4, $4, $4 #`
3. Submit a report printed on A4 paper at the beginning of the next lecture.
   - The report should include a block diagram, a source code in Verilog HDL, and obtained waveforms of your design.

Exploiting Instruction Level parallelism (ILP)

- A superscalar processor has to handle some flows efficiently to exploit ILP
  - Control flow
    - To execute n instructions per clock cycle, the processor has to fetch at least n instructions per cycle.
    - The main obstacles are branch instruction (BNE, BEQ)
    - Another obstacle is instruction cache
  - Register data flow
  - Memory data flow

Why do branch instructions degrade IPC?

- The branch taken / untaken is determined in execution stage of the branch.
- The conservative approach of stalling instruction fetch until the branch direction is determined.

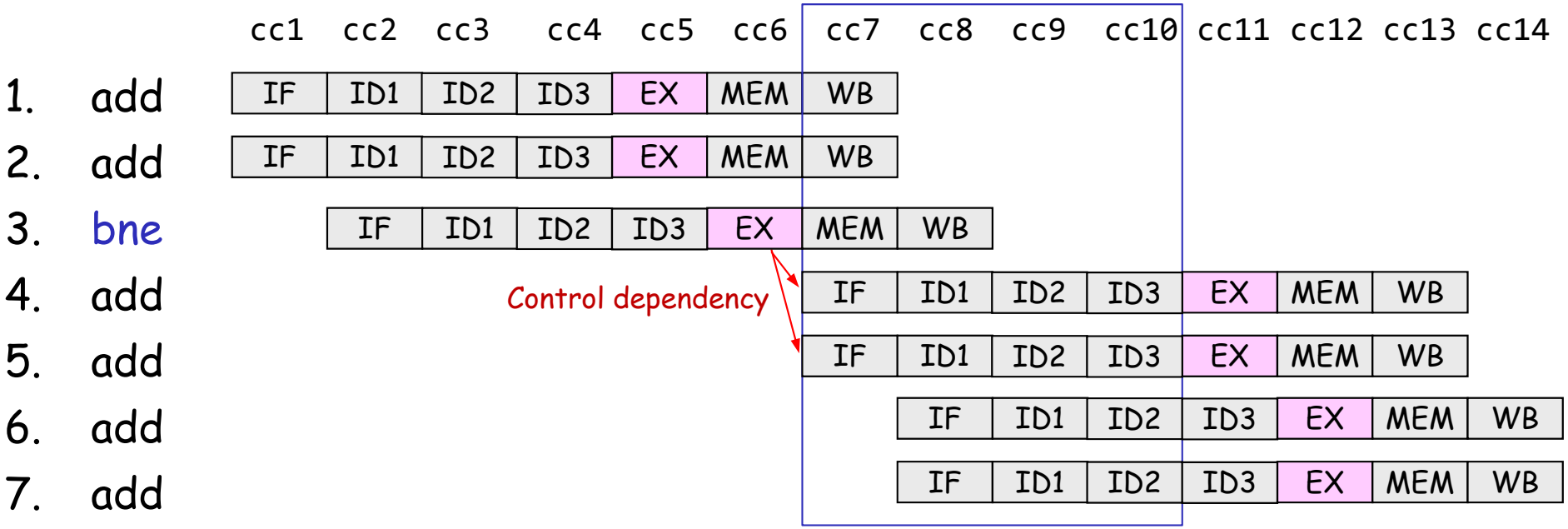


2-way superscalar processor executing instruction sequence with a branch

Note that because of a branch instruction, only one instruction is executed in cc4 and no instructions are executed in CC6 and CC7. This reduces the IPS.

Deeper pipeline

- In conservative approach, IPC degradation will be significant by deeper pipeline

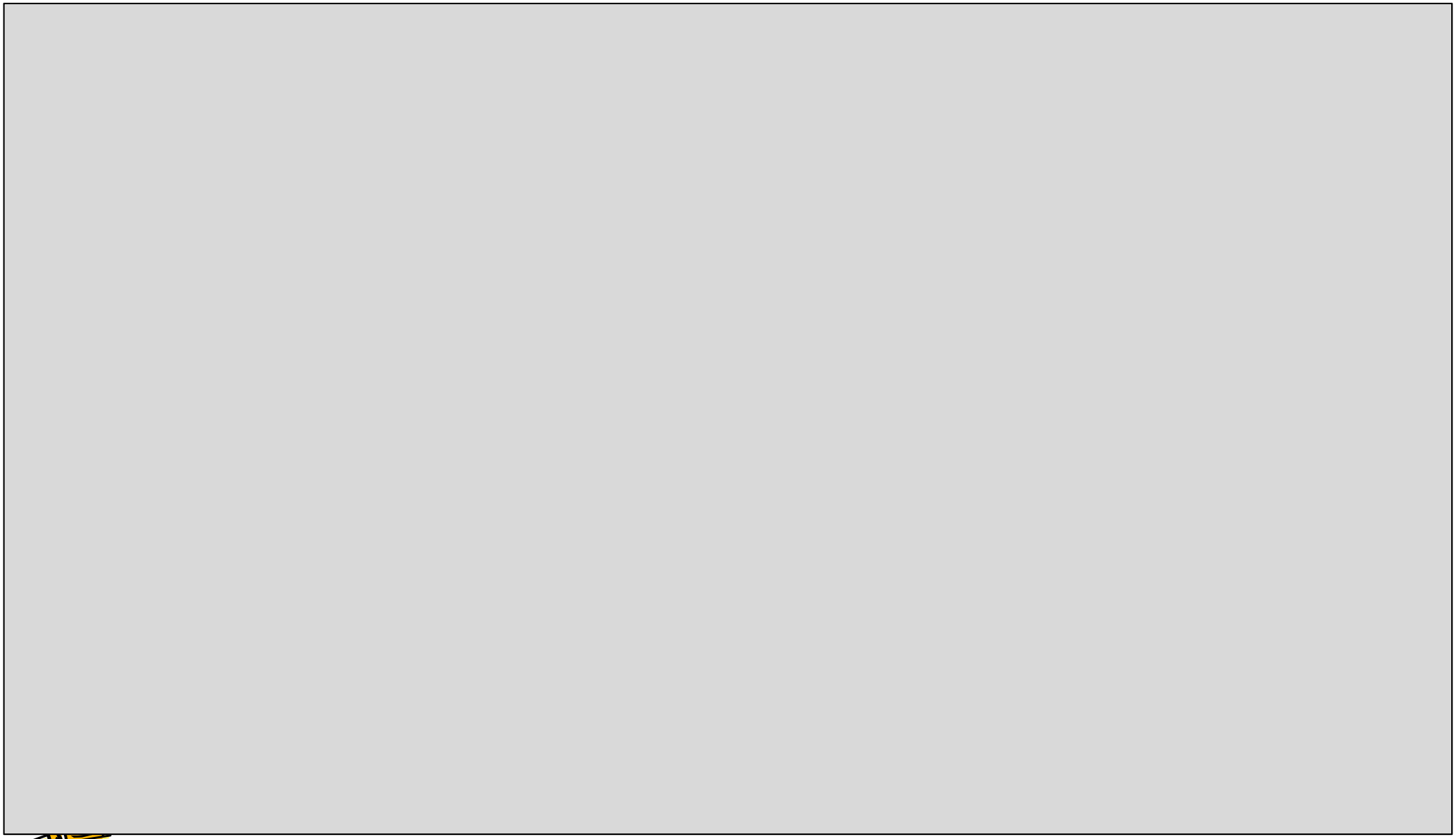


2-way superscalar adopting deeper pipeline executing instruction sequence with a branch

Branch predictor

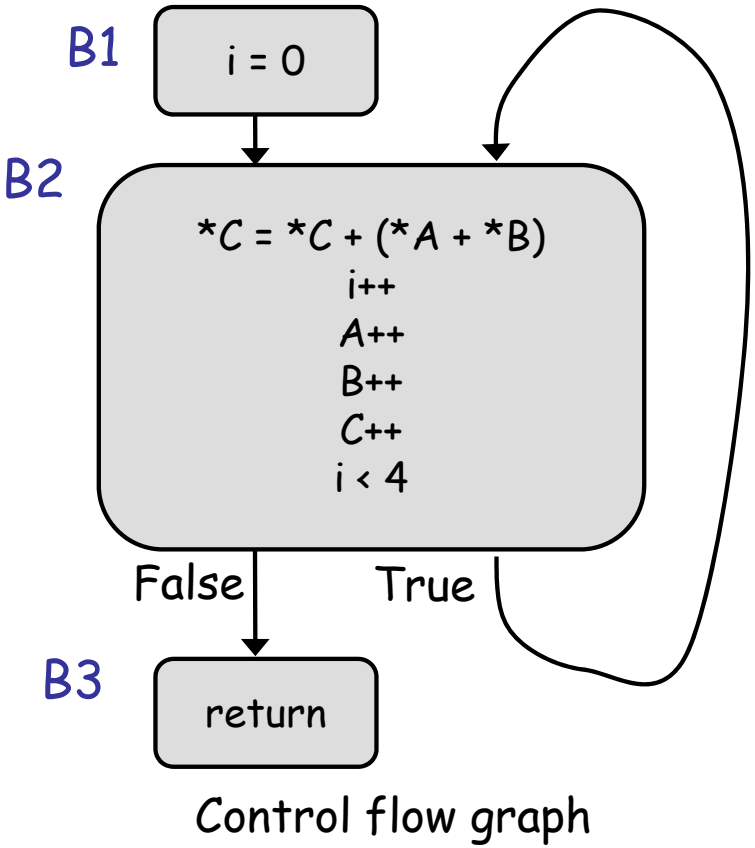
- A branch predictor is a digital circuit that tries to guess or predict which way (taken or untaken) a branch will go before this is known definitively.
  - A random predictor will achieve about a 50% hit rate because the prediction output is 1 or 0.
  - Let's guess the accuracy. What is the accuracy of typical branch predictors for high-performance commercial processors?

Prediction Accuracy of weather forecasts

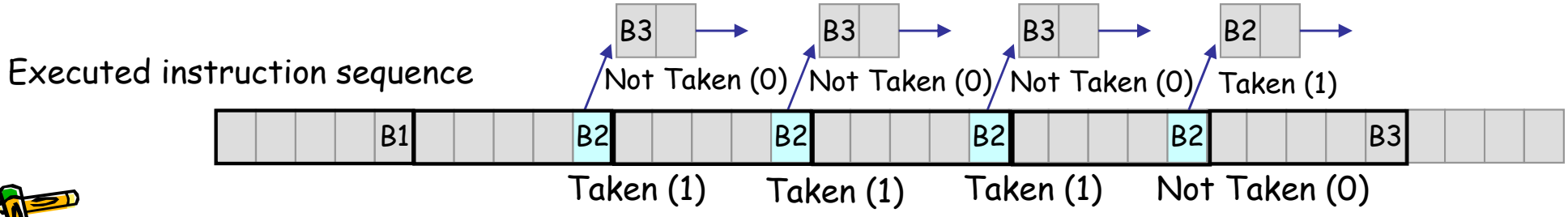


Sample program: vector add

```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
  for(i=0; i<VSIZE; i++)
    C[i] += (A[i] + B[i]);
}
```



Control flow graph

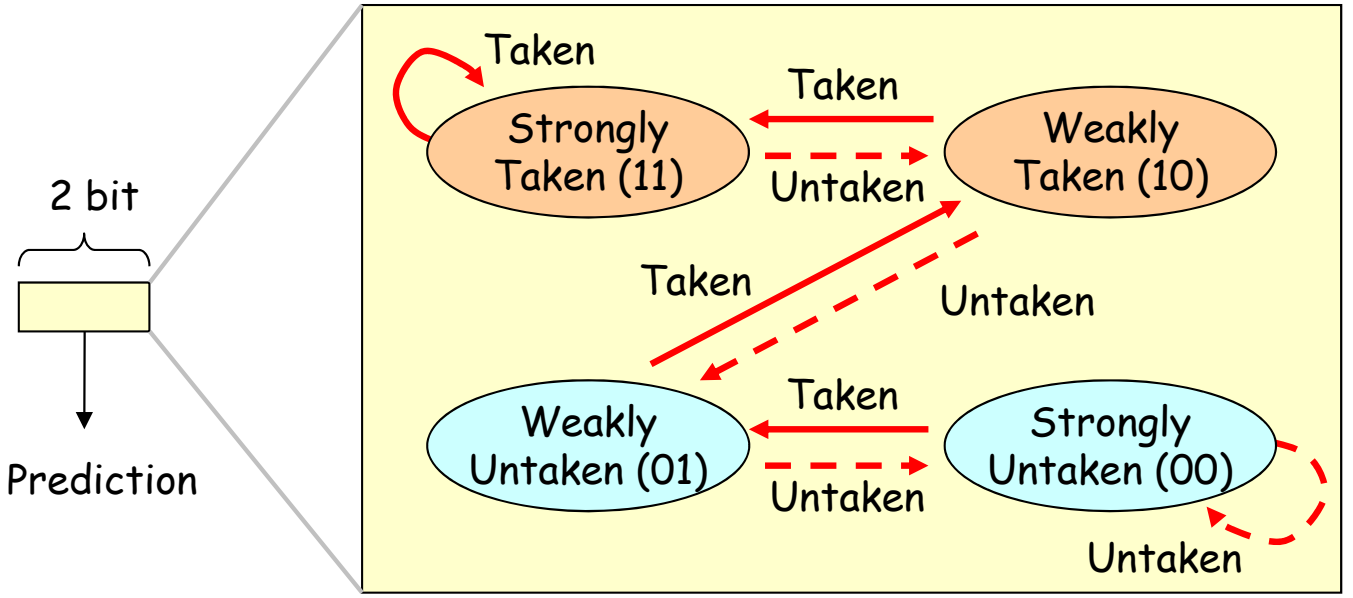


Simple branch predictor: Branch Always

- How to predict
  - It always predicts as 1.
- How to update
  - Nothing cause it does not use any memory.

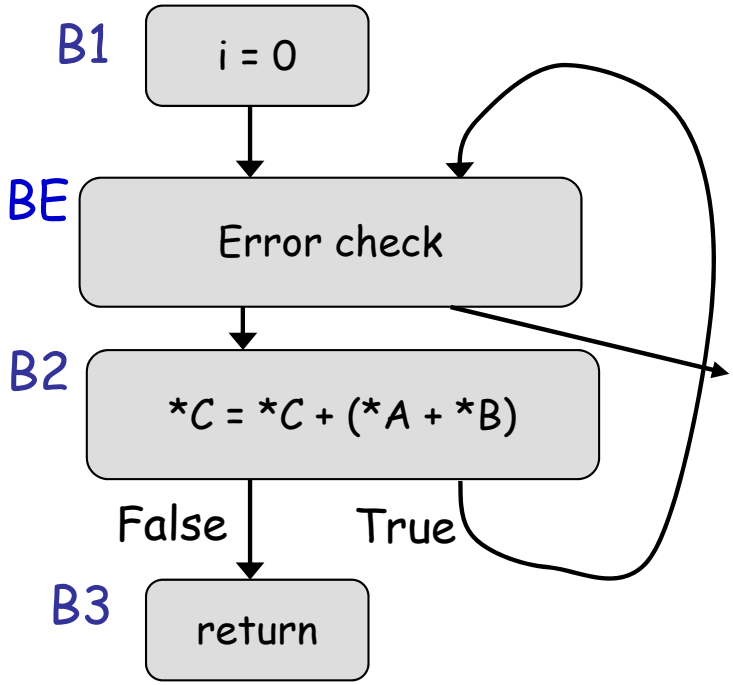
Simple branch predictor: 2bit counter

- It uses two bit register or a counter.
- Hot to predict
  - It predicts as 1 if the MSB of the register is one, otherwise predicts as 0.
- How to update the register
  - If the branch outcome is taken and the value is not 3, then increment the register.
  - If the branch outcome is untaken and the value is not 0, then decrement the register.



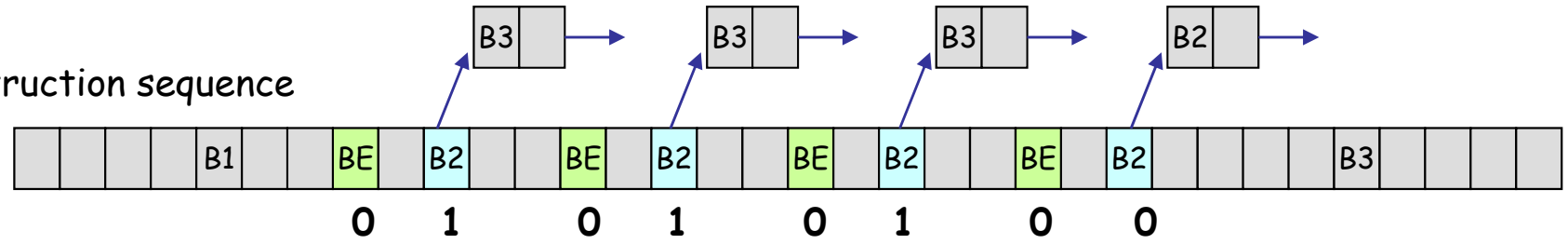
Sample program: vector add with two branches

```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
  for(i=0; i<VSIZE; i++) {
    if(A[i]<0) error_routine();
    C[i] += (A[i] + B[i]);
  }
}
```



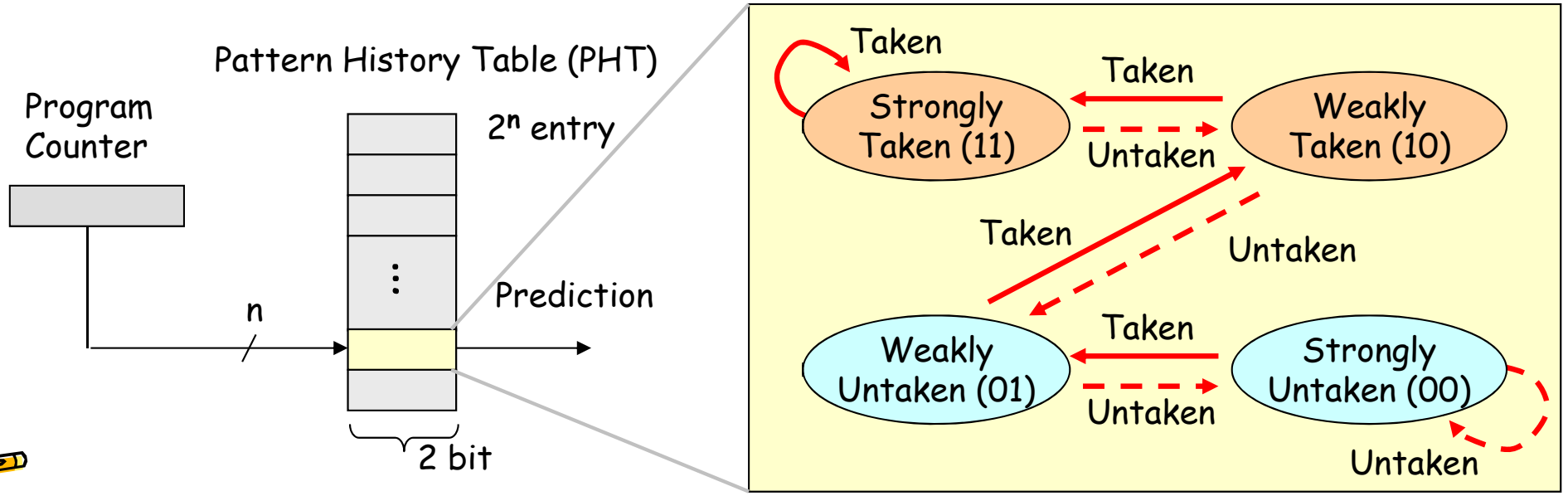
Control flow graph

Executed instruction sequence



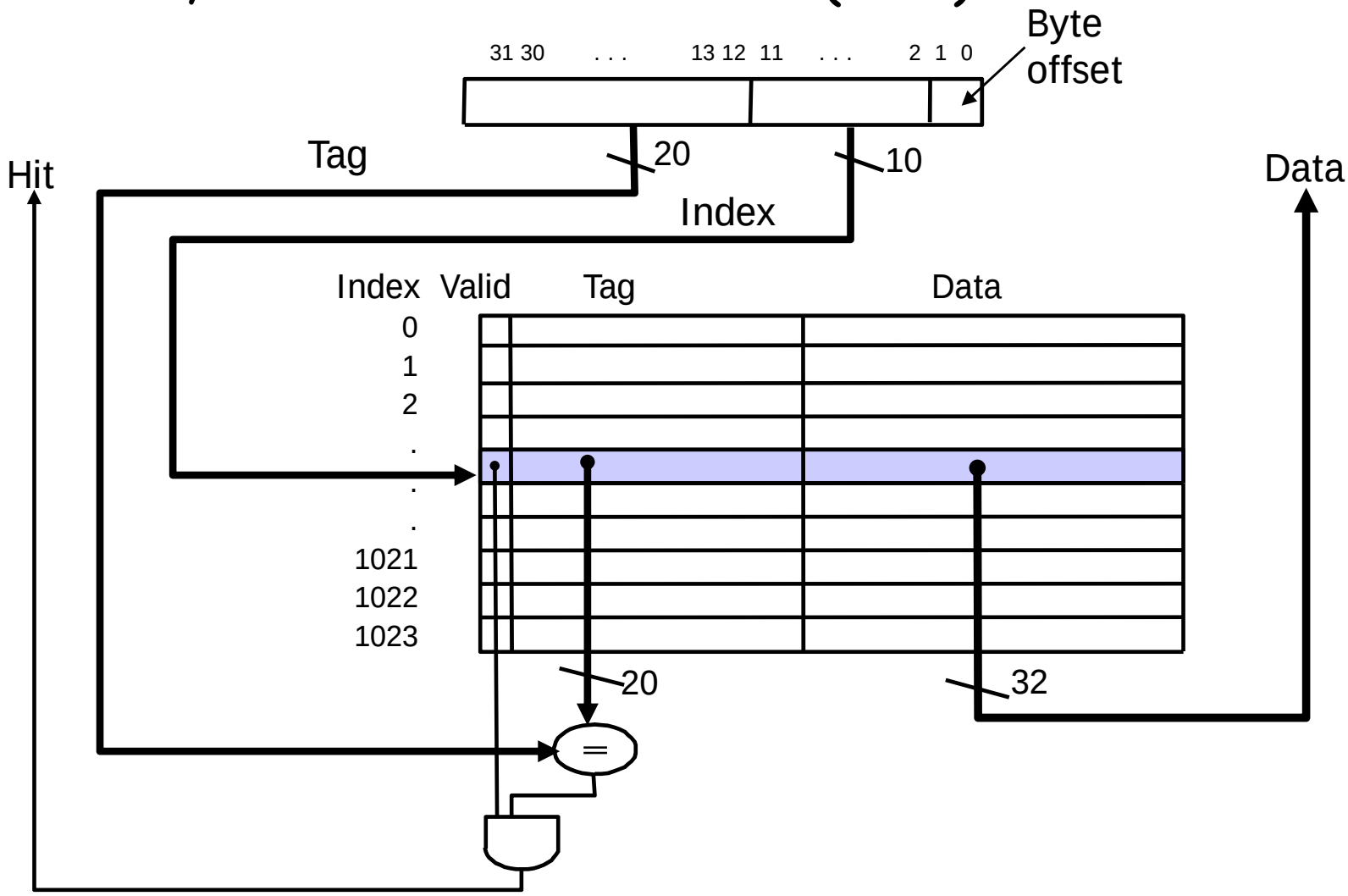
Simple branch predictor: bimodal

- Program has many branch instructions. The behavior may depend on each branch. Use one counter for one branch instruction
- How to predict
  - Select one counter using PC, then it predicts 1 if the MSB of the register is one, otherwise predicts 0.
- How to update
  - Select one counter using PC, then update the counter same manner as 2bit counter.



MIPS Direct Mapped Cache Example

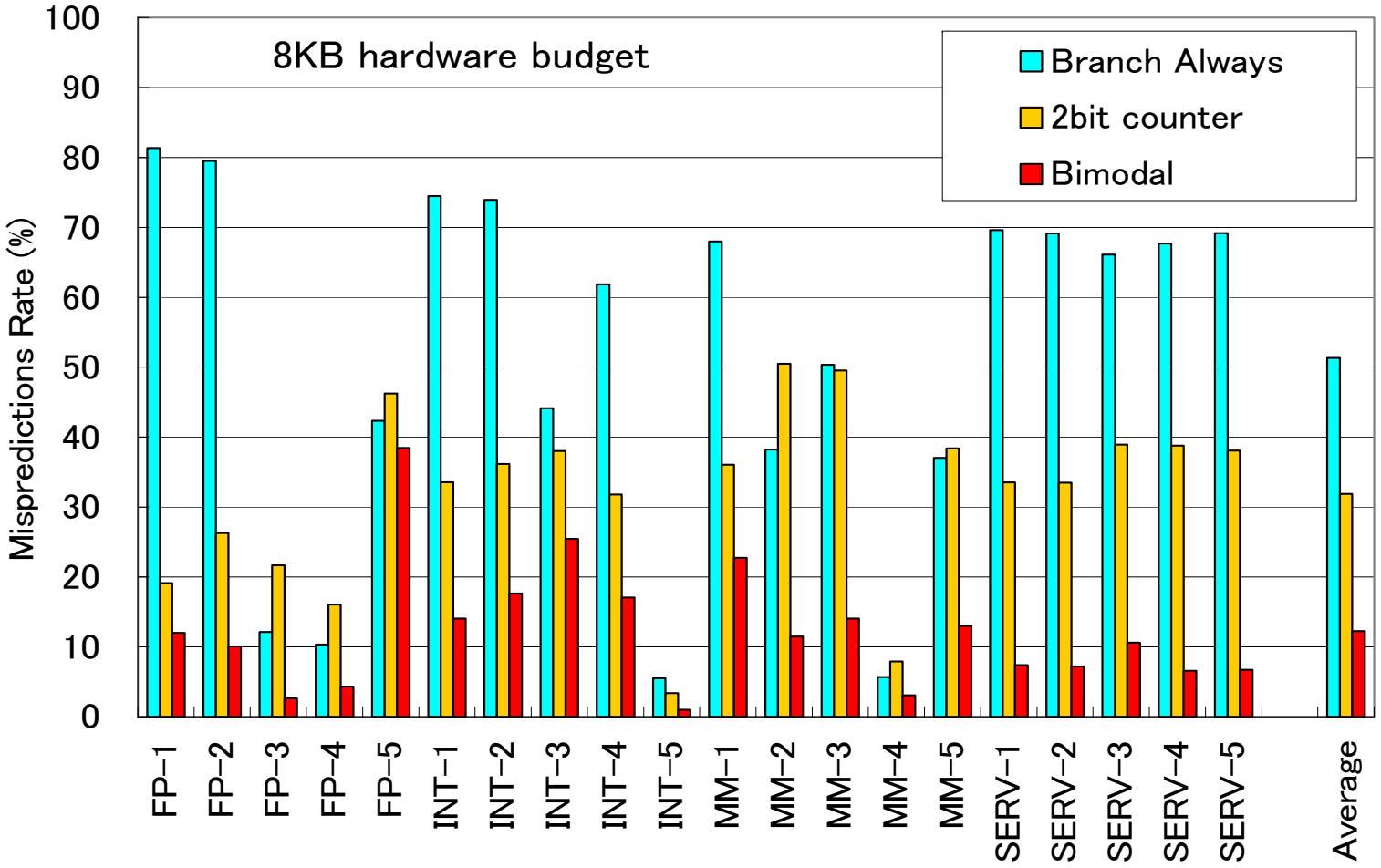
- One word/block, cache size = 1K words (4KB)



What kind of locality are we taking advantage of?

Prediction accuracy of simple branch predictors

- The accuracy of branch always is about 50%.
- The accuracy of bimodal predictor of 4KB memory is about 88%.



Benchmark for CBP(2004) by Intel MRL and IEEE TC uARCH.