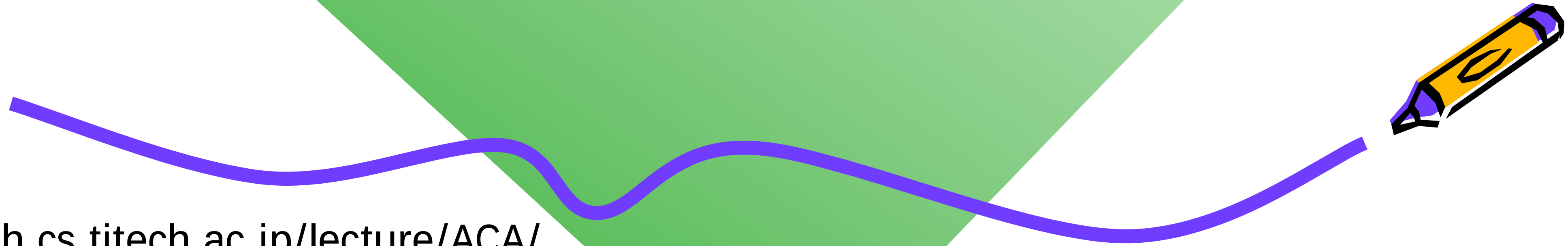
Fiscal Year 2019

Ver. 2019-12-23a

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

Advanced Computer Architecture

6. Instruction Level Parallelism: Instruction Fetch and Branch Prediction

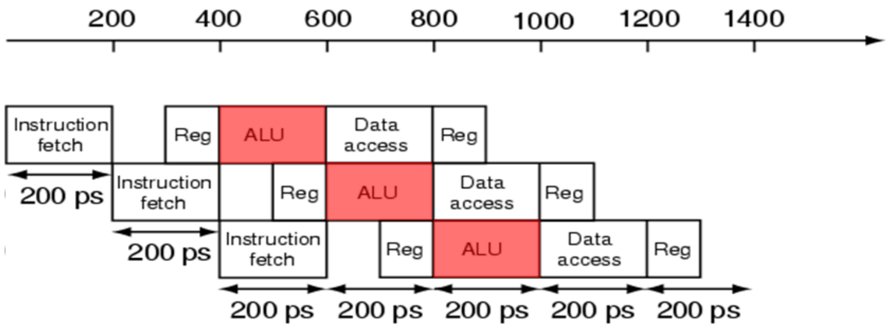


www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 13:20-14:50, Thr 13:20-14:50

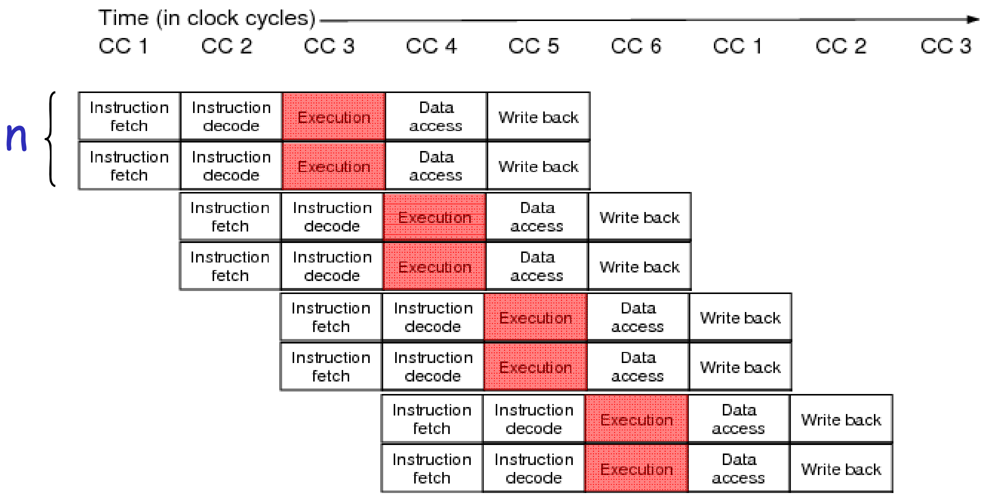
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

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

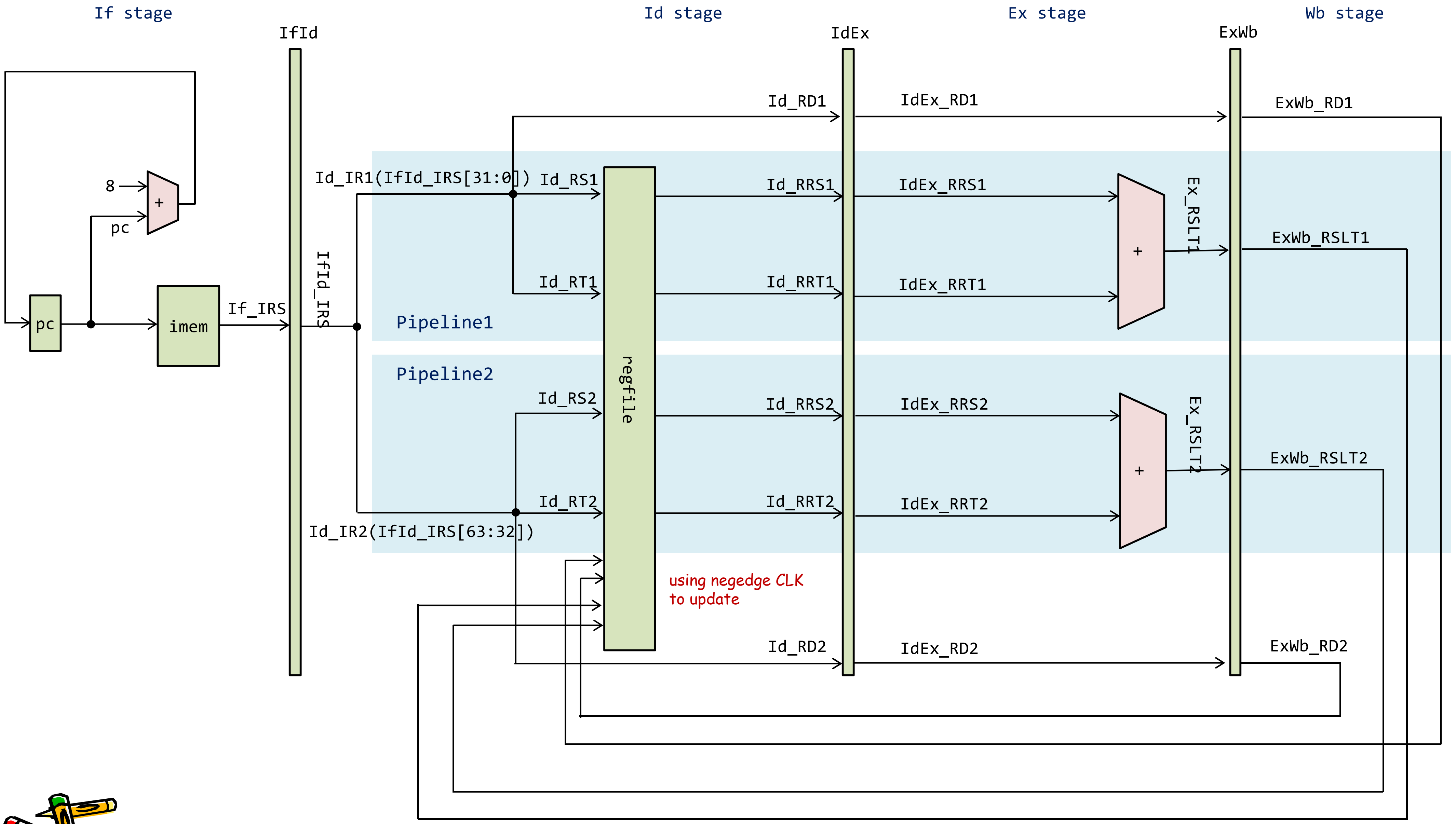


(a) pipeline diagram of scalar processor



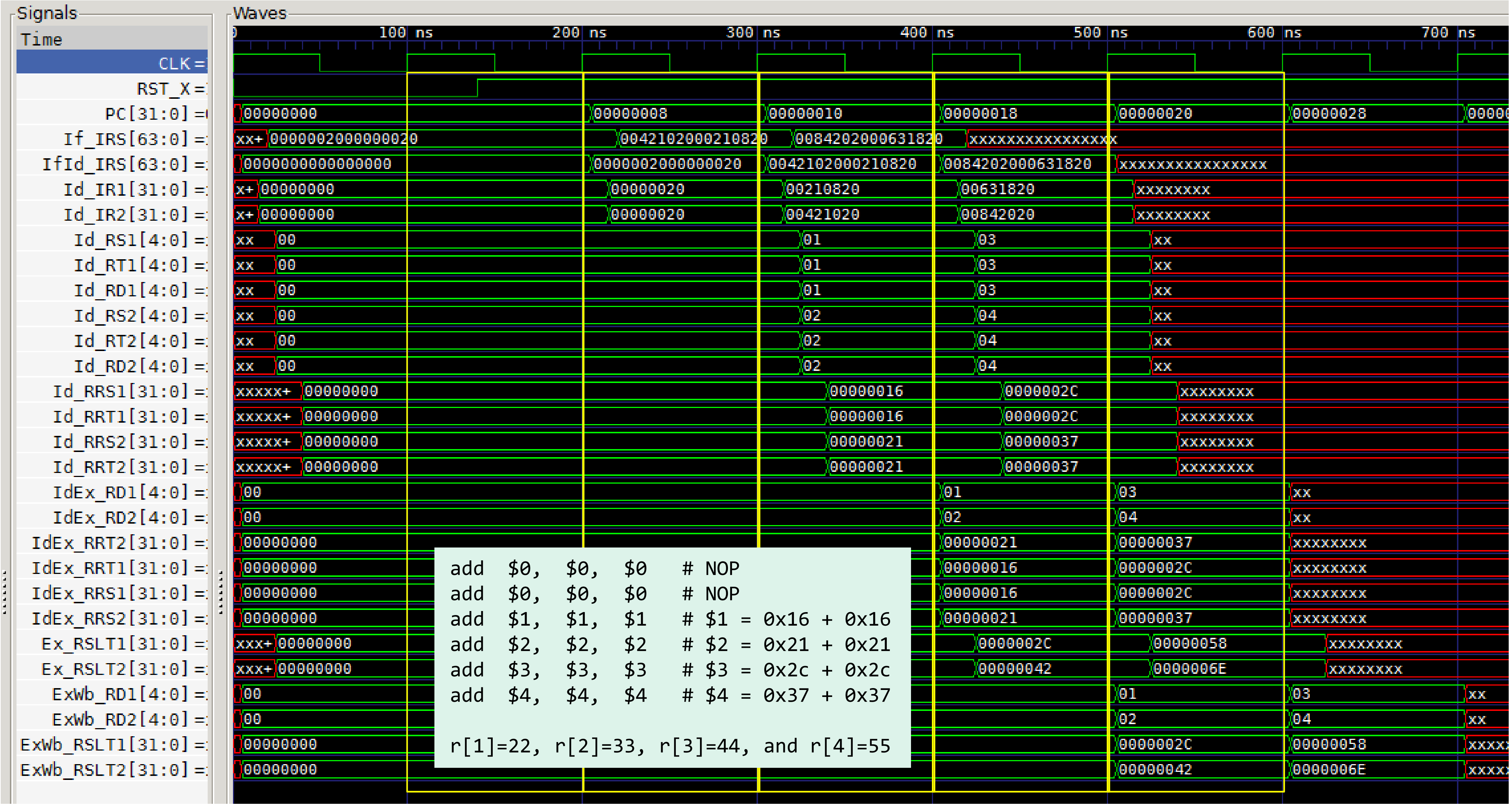
(b) pipeline diagram of 2-way superscalar processor

A four stage pipelined 2-way superscalar processor supporting ADD which does not adopt data forwarding (proc10, Assignment 5)



using negedge CLK to update

Waveform of Proc10


Signals
Waves
Time
0 100 ns 200 ns 300 ns 400 ns 500 ns 600 ns 700 ns
CLK =
RST_X =
PC[31:0] = 00000000 00000008 00000010 00000018 00000020 00000028 0000
If_IRS[63:0] = xx+ 0000002000000020 0042102000210820 0084202000631820 xxxxxxxxxxxxxxxx
IfId_IRS[63:0] = 0000000000000000 0000002000000020 0042102000210820 0084202000631820 xxxxxxxxxxxxxxxx
Id_IR1[31:0] = x+ 00000000 00000020 00210820 00631820 xxxxxxxx
Id_IR2[31:0] = x+ 00000000 00000020 00421020 00842020 xxxxxxxx
Id_RS1[4:0] = xx 00 01 03 xx
Id_RT1[4:0] = xx 00 01 03 xx
Id_RD1[4:0] = xx 00 01 03 xx
Id_RS2[4:0] = xx 00 02 04 xx
Id_RT2[4:0] = xx 00 02 04 xx
Id_RD2[4:0] = xx 00 02 04 xx
Id_RRS1[31:0] = xxxxx+ 00000000 00000016 0000002C xxxxxxxx
Id_RRT1[31:0] = xxxxx+ 00000000 00000016 0000002C xxxxxxxx
Id_RRS2[31:0] = xxxxx+ 00000000 00000021 00000037 xxxxxxxx
Id_RRT2[31:0] = xxxxx+ 00000000 00000021 00000037 xxxxxxxx
IdEx_RD1[4:0] = 00 01 03 xx
IdEx_RD2[4:0] = 00 02 04 xx
IdEx_RRT2[31:0] = 00000000 00000021 00000037 xxxxxxxx
IdEx_RRT1[31:0] = 00000000 00000016 0000002C xxxxxxxx
IdEx_RRS1[31:0] = 00000000 00000016 0000002C xxxxxxxx
IdEx_RRS2[31:0] = 00000000 00000021 00000037 xxxxxxxx
Ex_RSLT1[31:0] = xxx+ 00000000 0000002C 00000058 xxxxxxxx
Ex_RSLT2[31:0] = xxx+ 00000000 00000042 0000006E xxxxxxxx
ExWb_RD1[4:0] = 00 01 03 xx
ExWb_RD2[4:0] = 00 02 04 xx
ExWb_RSLT1[31:0] = 00000000 0000002C 00000058 xxxx
ExWb_RSLT2[31:0] = 00000000 00000042 0000006E xxxx

```
add  $0,  $0,  $0   # NOP
add  $0,  $0,  $0   # NOP
add  $1,  $1,  $1   # $1 = 0x16 + 0x16
add  $2,  $2,  $2   # $2 = 0x21 + 0x21
add  $3,  $3,  $3   # $3 = 0x2c + 0x2c
add  $4,  $4,  $4   # $4 = 0x37 + 0x37

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

Exploiting Instruction Level parallelism (ILP)

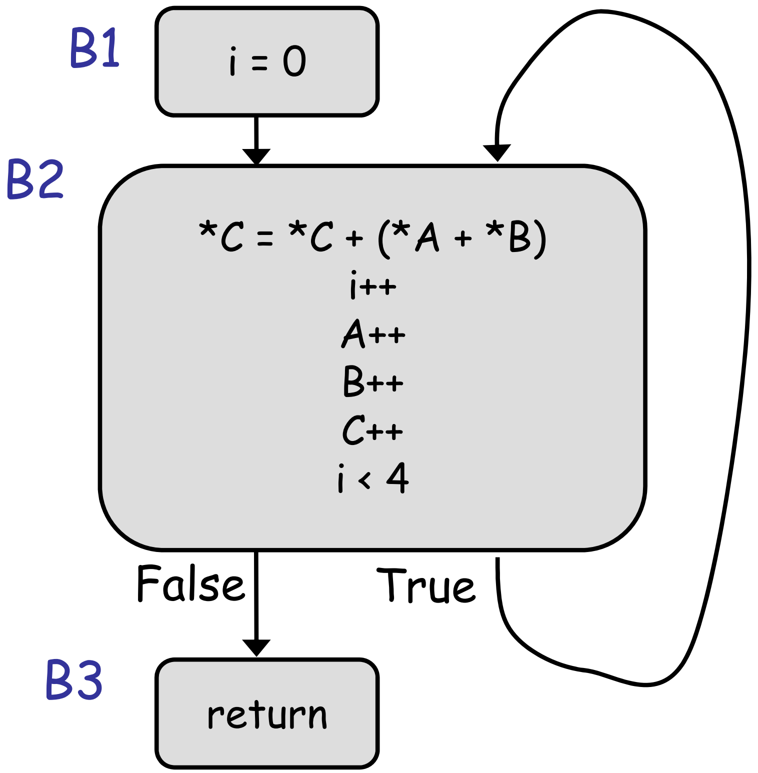
- A superscalar processor has to handle some flows efficiently to exploit ILP
    - Control flow
        - To execute *n* instructions per clock cycle, the processor has to fetch at least *n* instructions per cycle.
        - The main obstacles are branch instruction (BNE, BEQ)
        - Another obstacle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

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?

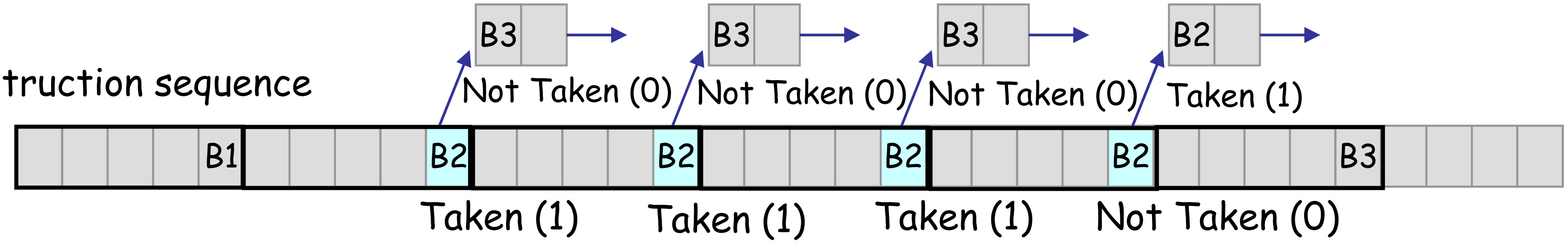
Sample program: vector add

```
#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

Executed instruction sequence

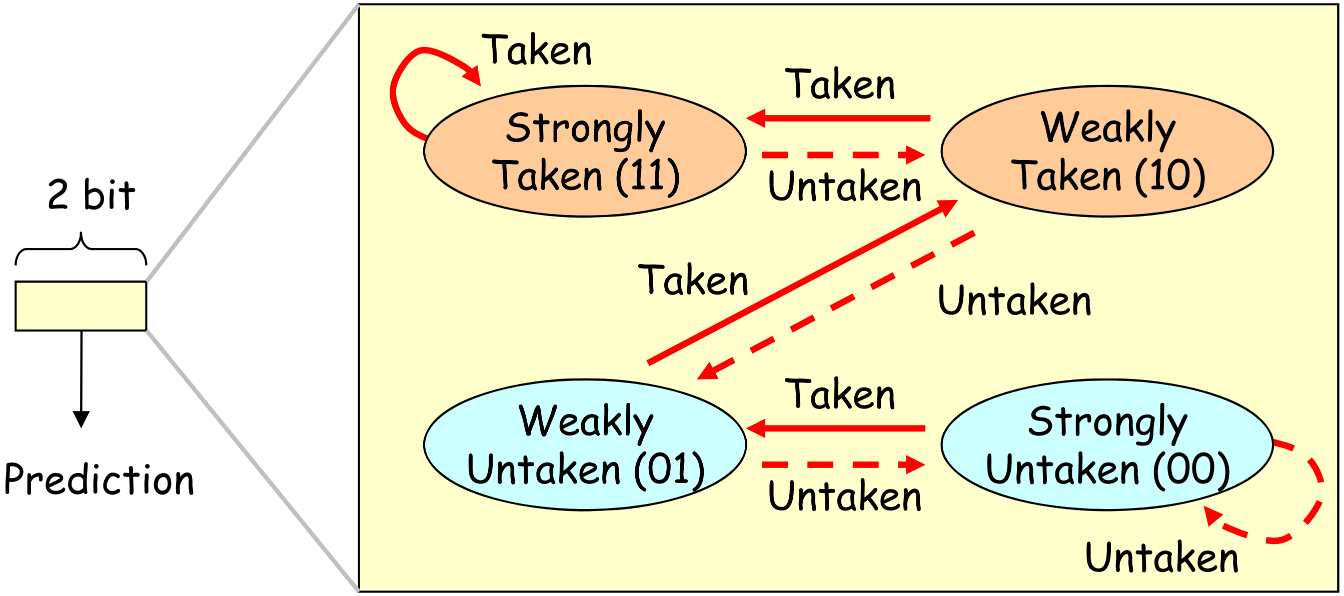


Simple branch predictor: Branch Always

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

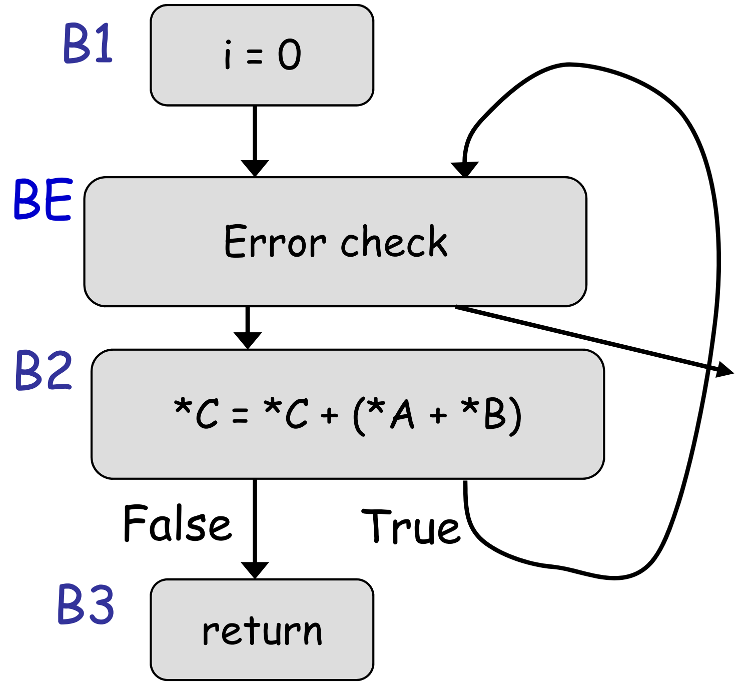
Simple branch predictor: 2bit counter

- 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.



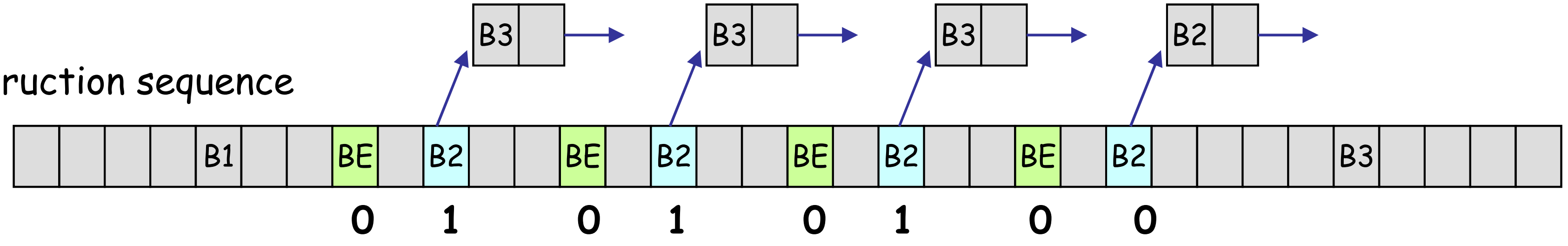
Sample program: vector add with two branches

```
#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]);
  }
}
```



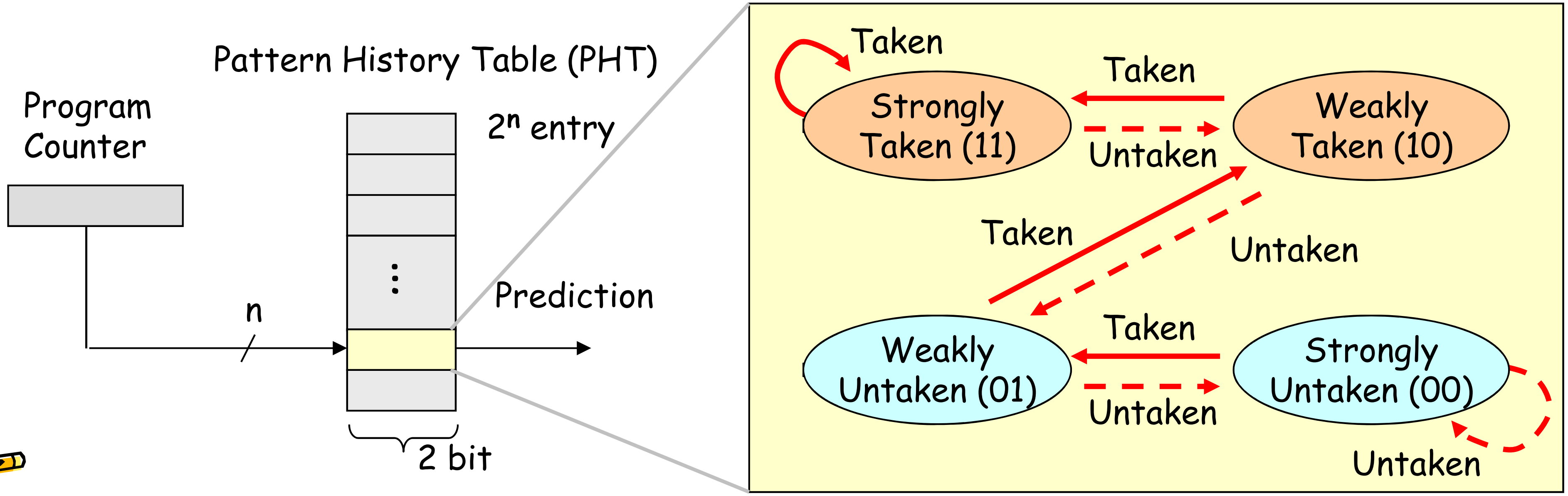
Control flow graph

Executed instruction sequence



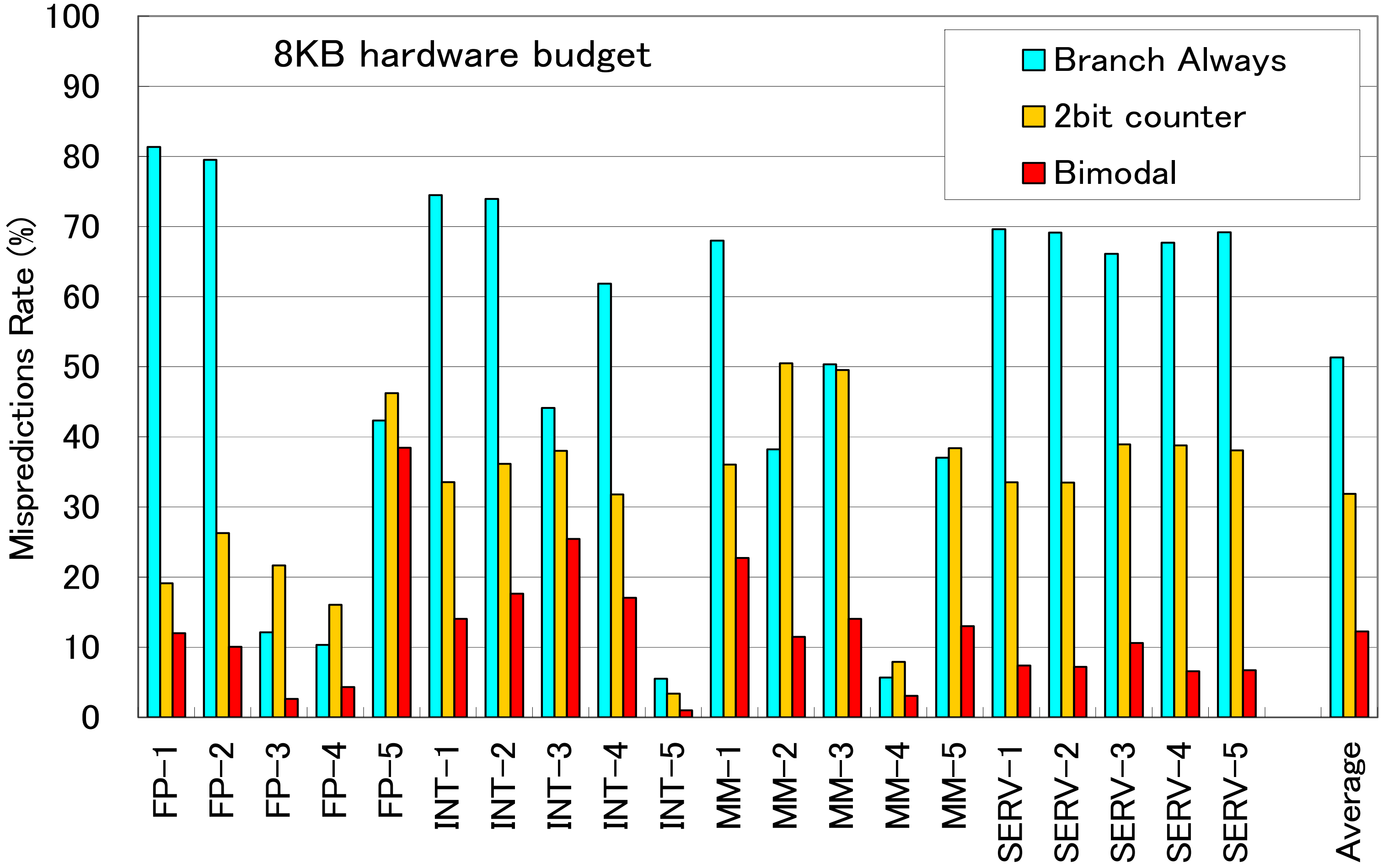
Simple branch predictor: bimodal

- 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.



Prediction accuracy of simple branch predictors

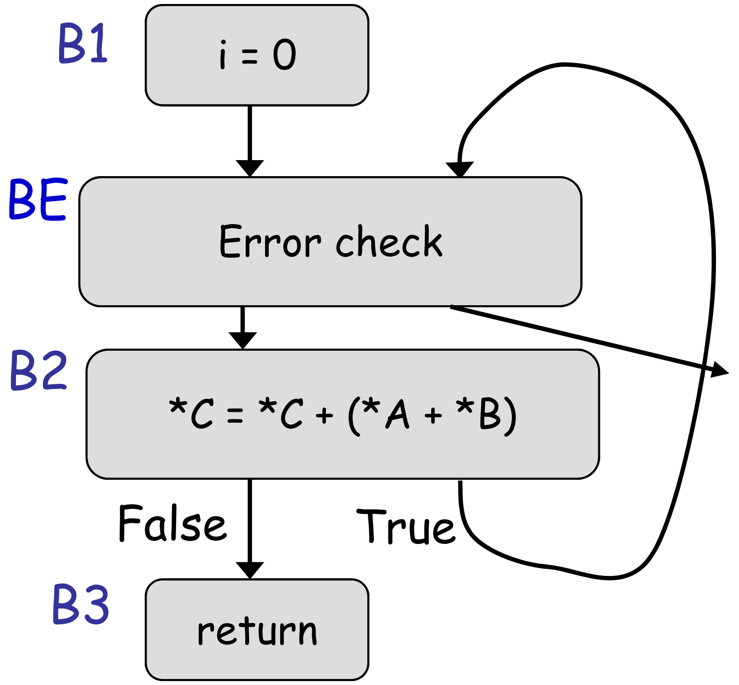
- The accuracy of branch always is about 50%.
- The accuracy of bimodal predictor of 4KB memory is about 88%.



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

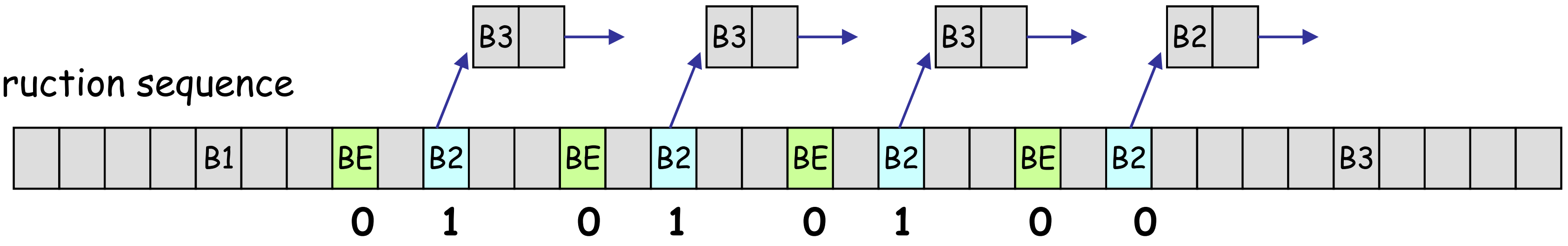
Sample program: vector add with two branches

```
#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]);
  }
}
```



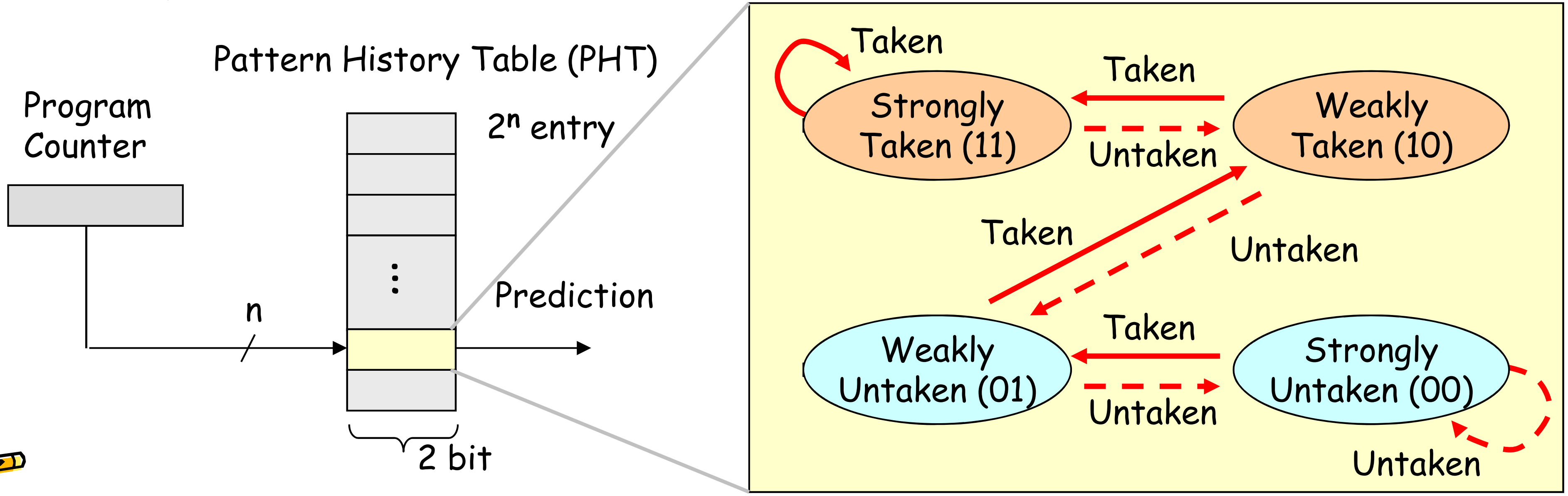
Control flow graph

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 in the same way as 2bit counter.



An innovation in branch predictors in 1993

- Using branch history
  - global branch history
  - local branch history
- 2-level branch predictor and Gshare

- *Assume predicting the sequence* 1110 1110 1110 1110 1110 …

```
11101110 ?
111011101 ?
1110111011 ?
11101110111 ?
111011101110 ?
```

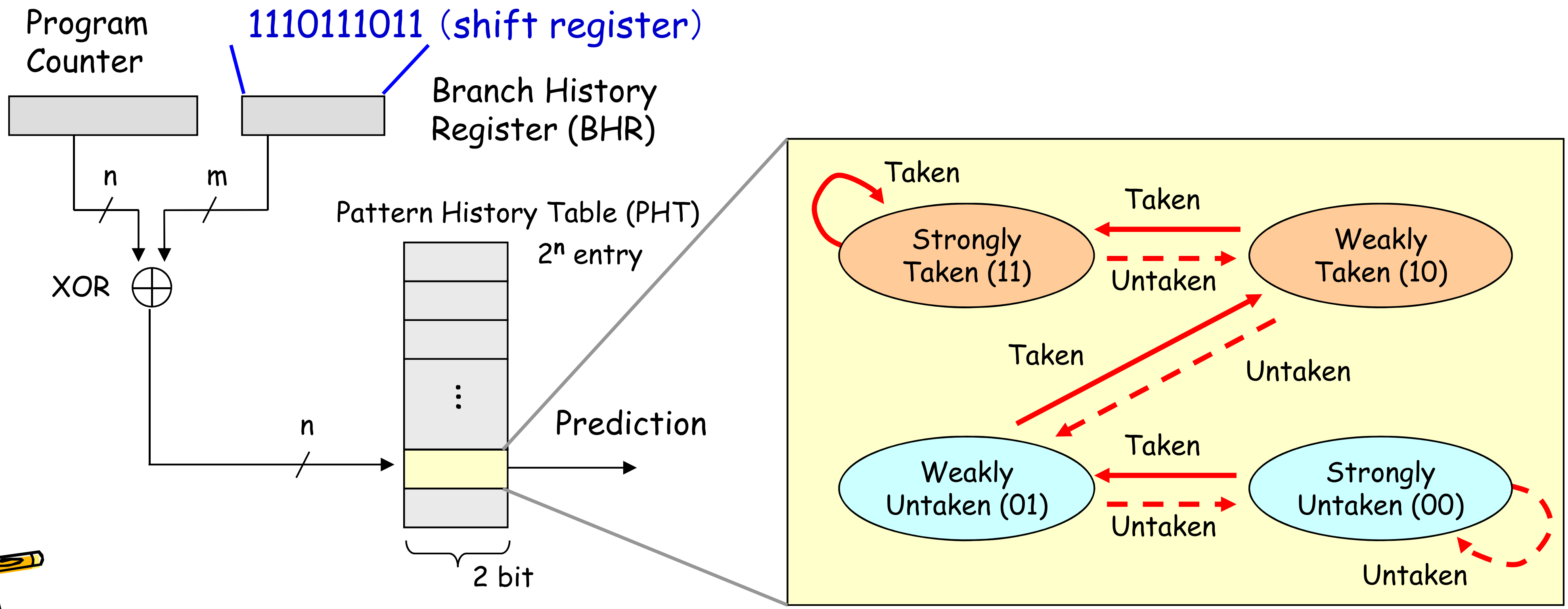
Recommended Reading

- Combining Branch Predictors
    - Scott McFarling, Digital Western Research Laboratory
    - WRL Technical Note TN-36, 1993

- A quote:
  "In this paper, we have presented two new methods for improving branch prediction performance. First, we showed that using the bit-wise exclusive OR of the global branch history and the branch address to access predictor counters results in better performance for a given counter array size."

Gshare (TR-DEC 1993)

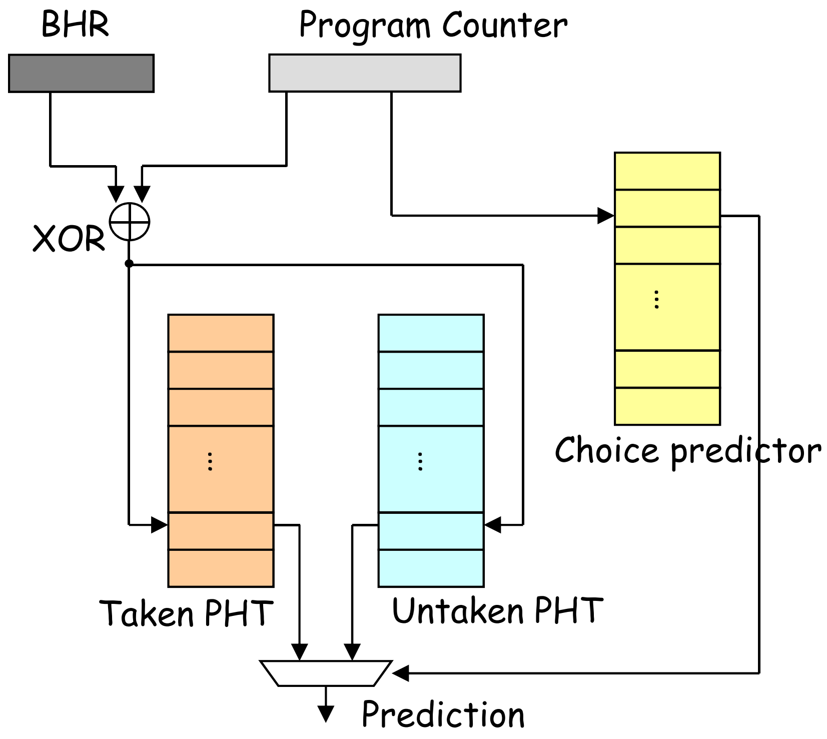
- How to predict
  - Using the exclusive OR of the global branch history and PC to access PHT, then MSB of the selected counter is the prediction.
- How to update
  - Shifting BHR one bit left and update LSB by branch outcome in IF stage.
  - Update the used counter in the same way as 2BC in WB stage.



Exercise: how to update PHT and BHR of Gshare

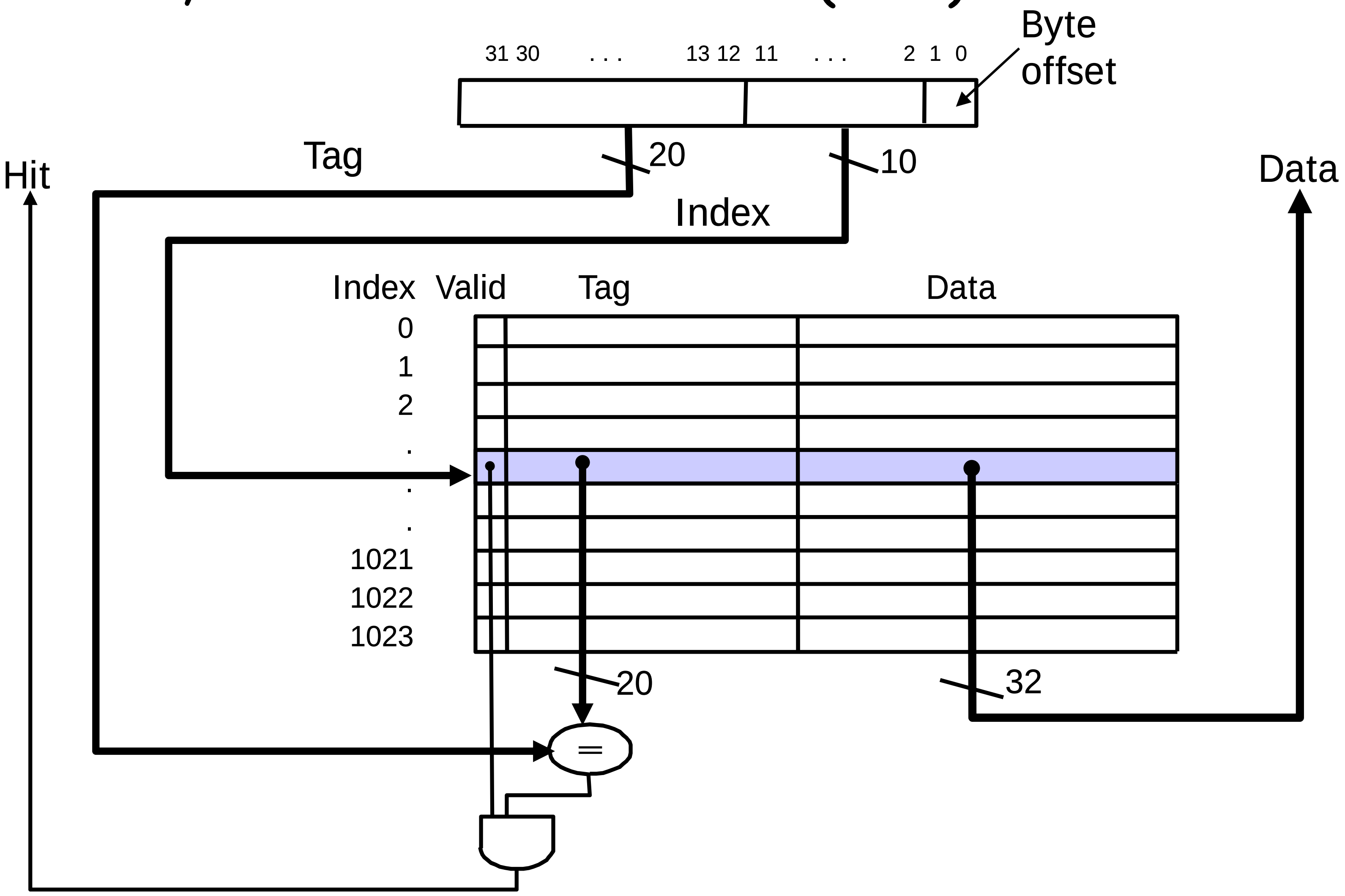
Bi-Mode (MICRO 1997)

- A choice predictor (bimodal) is used as a meta-predictor
- How to predict
  - Like Gshare, both of Taken PHT and Untaken PHT make two predictions.
  - Select one among them by the choice predictor which tracks the global bias of a branch.
- How to update
  - The used PHT is updated in the same way as 2BC.
  - Choice predictor is update in the same way as bimodal



MIPS Direct Mapped Cache Example

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



What kind of locality are we taking advantage of?

YAGS, Yet Another Global Scheme (MICRO 1998)

- Using two tagged PHTs
- When a PHT miss, choice PHT makes a prediction.

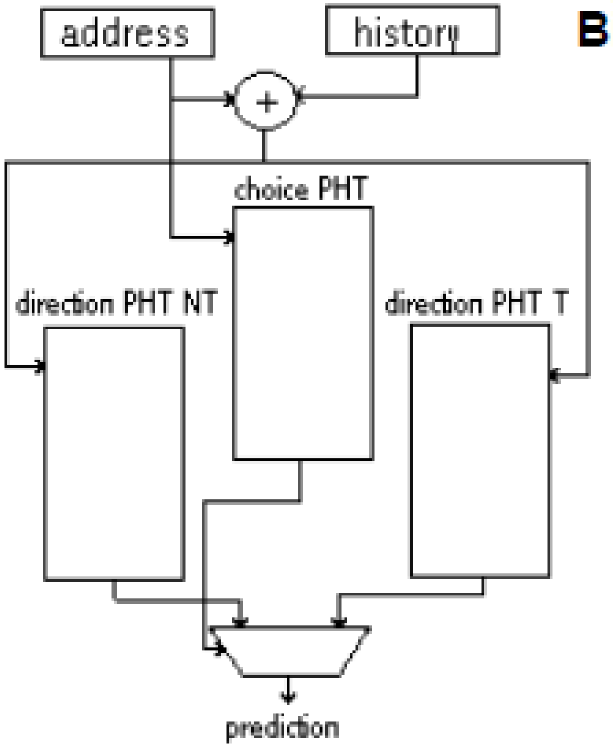


Figure 3. Bi-Mode

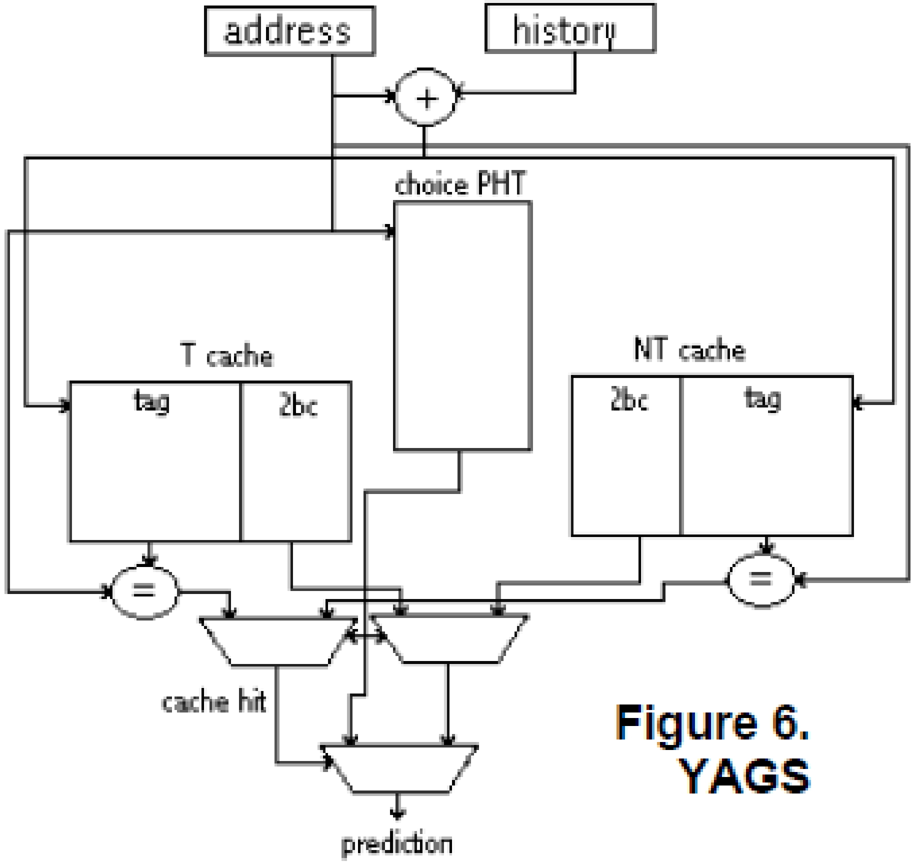
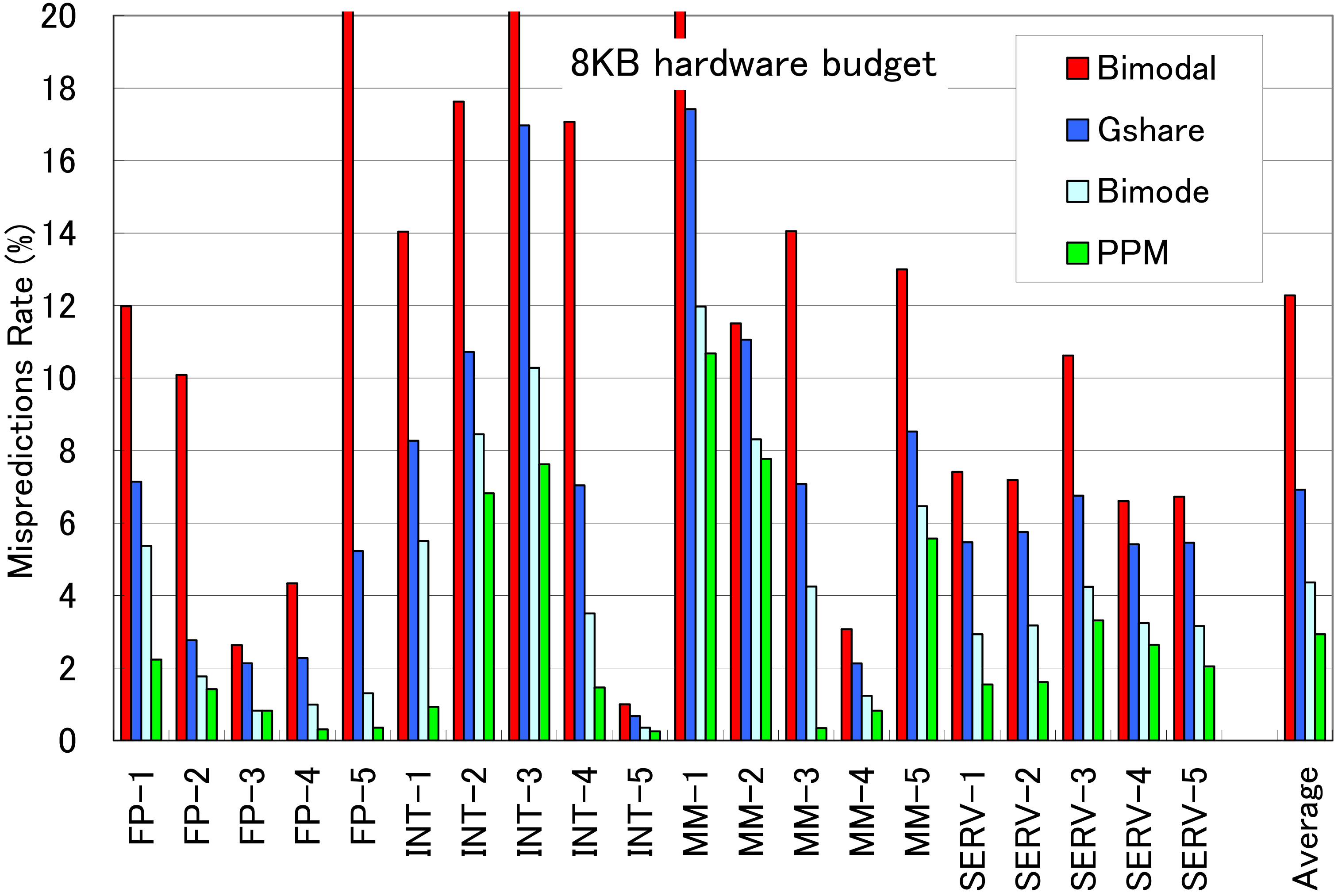


Figure 6. YAGS

From YAGS paper

Prediction accuracy

- The accuracy of 4KB Gshare is about 93%.
- The accuracy of 4KB PPM is about 97%.



Assignment 6

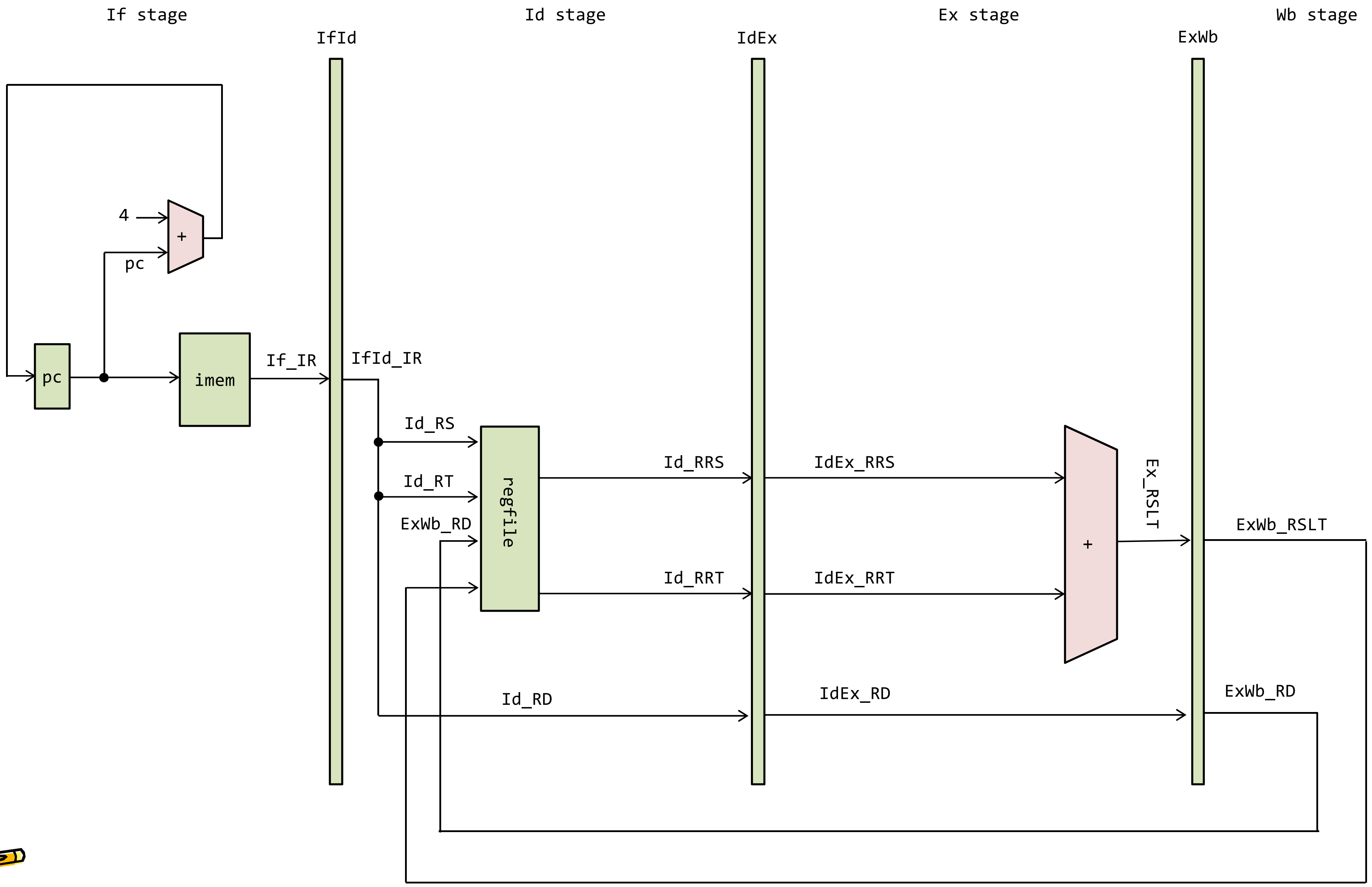
1. Design a four stage pipelined scalar processor supporting MIPS add and bne instruction in Verilog HDL. Please download proc06.v and proc07.v from the support page and refer it.
2. Verify the behavior of designed processor using following assembly code.

```
p.imem.mem[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.imem.mem[1] = {6'h0, 5'd5, 5'd1, 5'd5, 5'd0, 6'h20};  // L1: add  $5, $5, $1
p.imem.mem[2] = {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.imem.mem[3] = {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.imem.mem[4] = {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.imem.mem[5] = {6'h5, 5'd4, 5'd5, 16'hfffb};           //     bne  $4, $5, L1
p.imem.mem[6] = {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.imem.mem[7] = {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.imem.mem[8] = {6'h0, 5'd0, 5'd0, 5'd5, 5'd0, 6'h20};  //     add  $5, $0, $0
p.imem.mem[9] = {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.imem.mem[10]= {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.imem.mem[11]= {6'h5, 5'd2, 5'd0, 16'hfff5};           //     bne  $2, $0, L1
p.imem.mem[12]= {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.imem.mem[13]= {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20};  //     NOP
p.regfile.r[1] = 1; p.regfile.r[2] = 22; p.regfile.r[3] = 0; p.regfile.r[4] = 4; p.regfile.r[5] = 0;
```

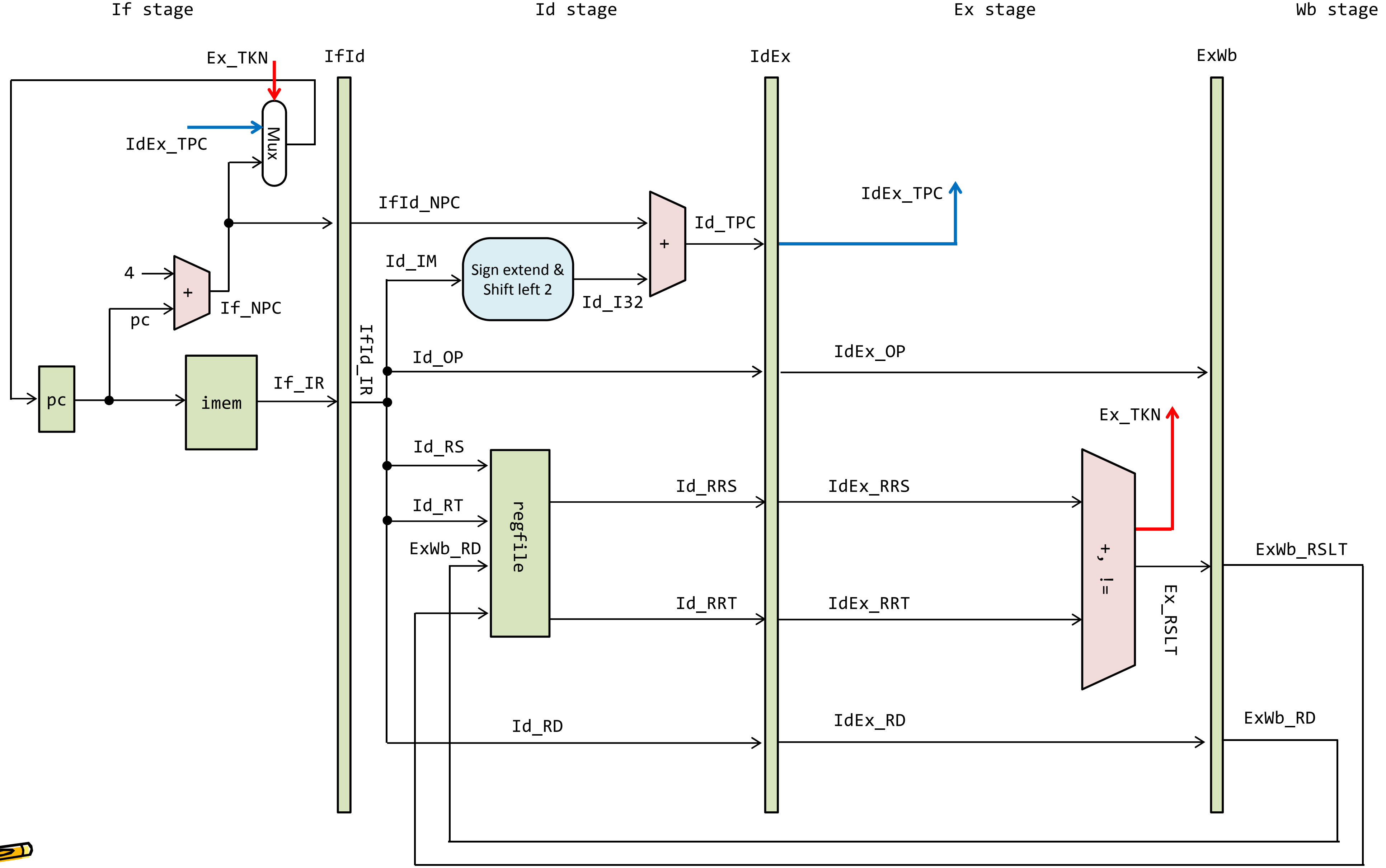
```
while(1){
  for(int i=1; i!=4; i++){

  }
}
```

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.

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



Four stage pipelined processor supporting ADD and BNE, which does not adopt data forwarding (proc08.v, Assignment 6)


If stage
Id stage
Ex stage
Wb stage
Ex_TKN
IfId
IdEx
ExWb
IdEx_TPC
Mux
IfId_NPC
Id_TPC
IdEx_TPC
+
Id_IM
Sign extend & Shift left 2
4
+
If_NPC
pc
Id_I32
IfId_IR
Id_OP
IdEx_OP
pc
imem
If_IR
Ex_TKN
Id_RS
Id_RRS
IdEx_RRS
Id_RT
regfile
ExWb_RD
+, !=
ExWb_RSLT
Ex_RSLT
Id_RRT
IdEx_RRT
Id_RD
IdEx_RD
ExWb_RD