Advanced Computer Architecture

—

Part I: General Purpose
SimpleScalar Practical Lab (II)

EPFL – I&C – LAP

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

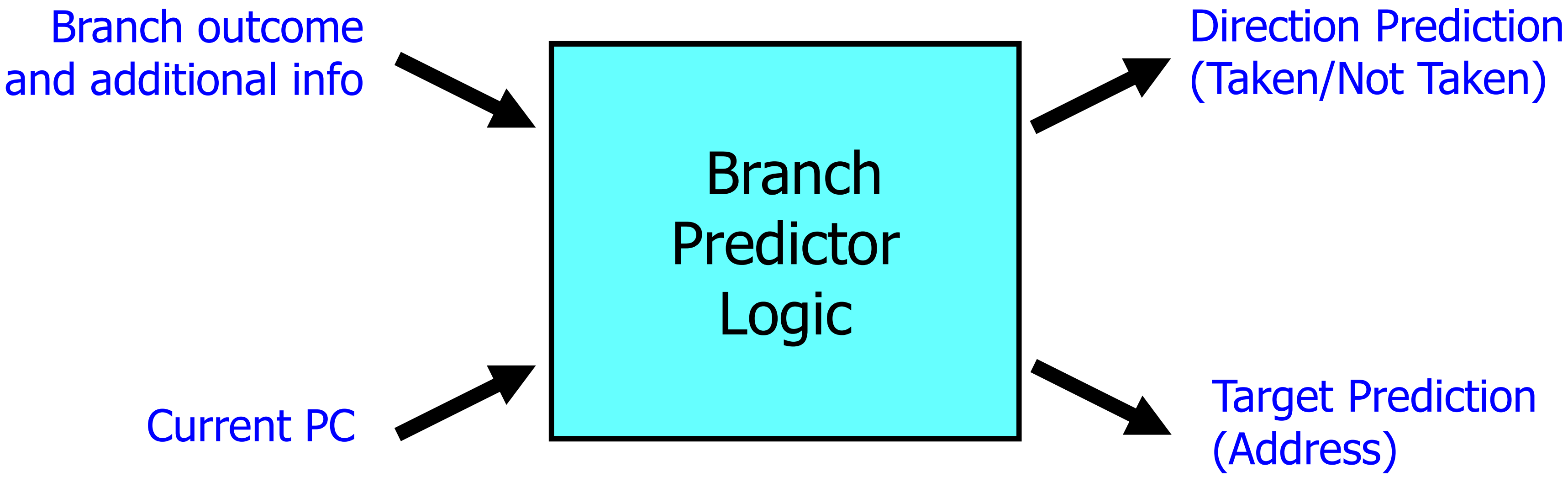
Branch Prediction

- Branches change program execution direction
  - Determine next instruction after branch execution
  - Stalls become expensive
- "Predict" if a branch is taken
  - Speculate program execution
  - Improve performance if you predict correctly
- Our goal is to study different branch prediction techniques and their effect on performance

Branch Prediction Types

- Perfect prediction
  - With oracles  or in simulation (remember simplescalar)
- Static prediction: Always taken/not taken
  - Simple hardware solution
  - Predict your branches when designing your hardware
- Dynamic prediction
  - Predict your branches in runtime
  - Learn from your execution history

Branch Prediction Operation



Dynamic Direction Prediction
Bimodal Predictor

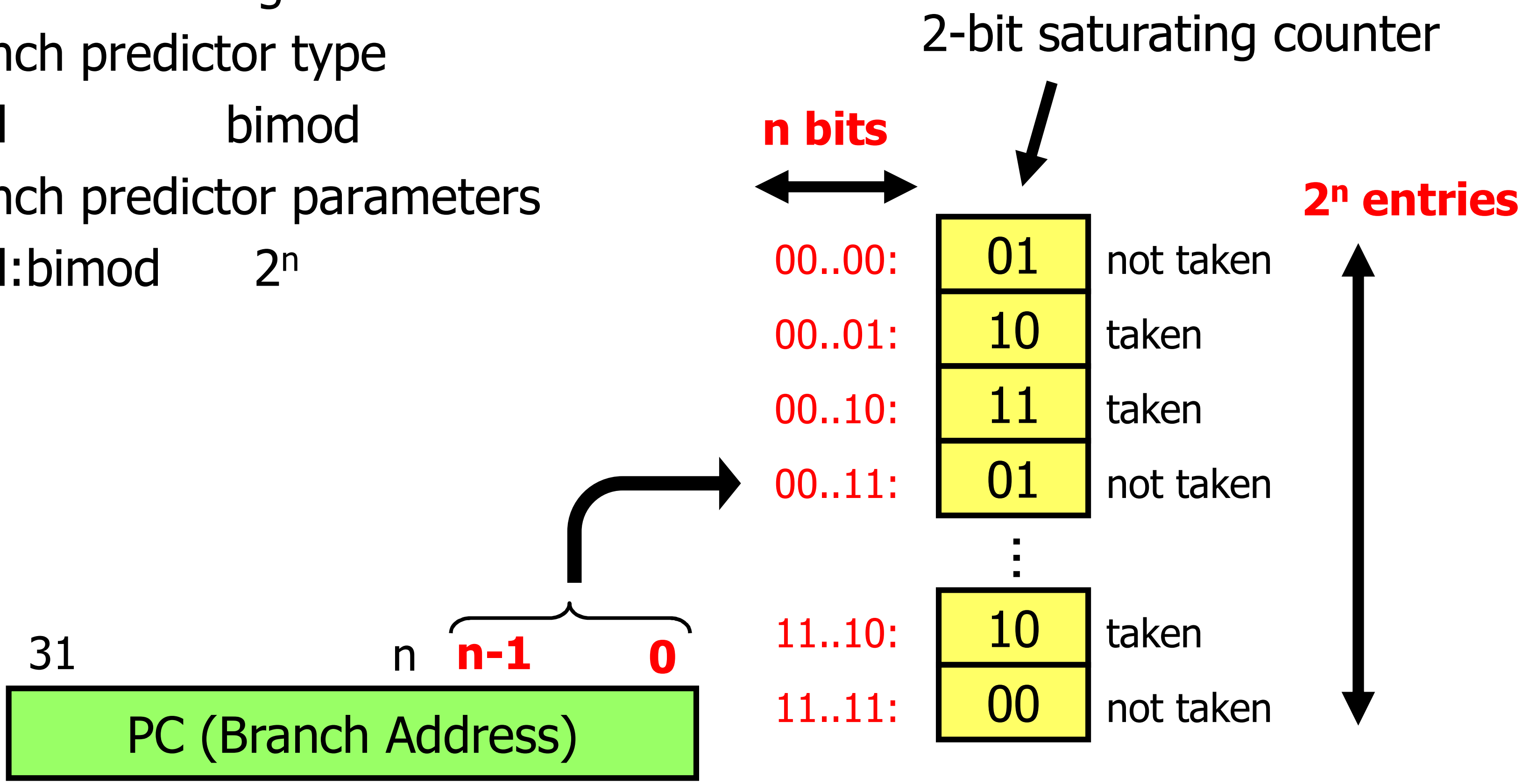
Simplescalar configuration

\# branch predictor type

-bpred bimod

\# branch predictor parameters

-bpred:bimod 2^n



Dynamic Direction Prediction
2-level Adaptive Predictor

Simplescalar configuration

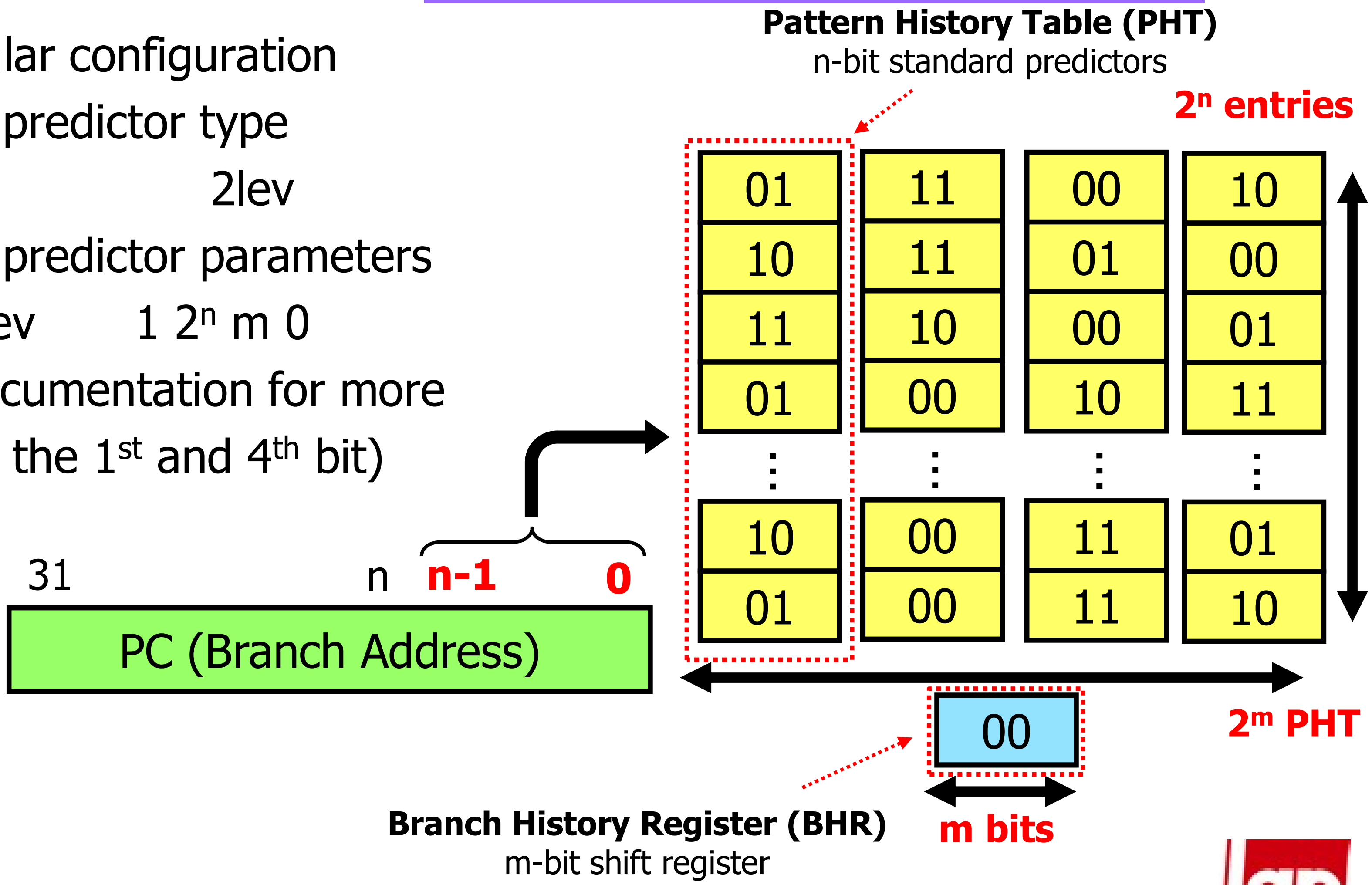
\# branch predictor type

-bpred 2lev

\# branch predictor parameters

-bpred:2lev 1 2^n m 0

(check documentation for more details on the 1st and 4th bit)



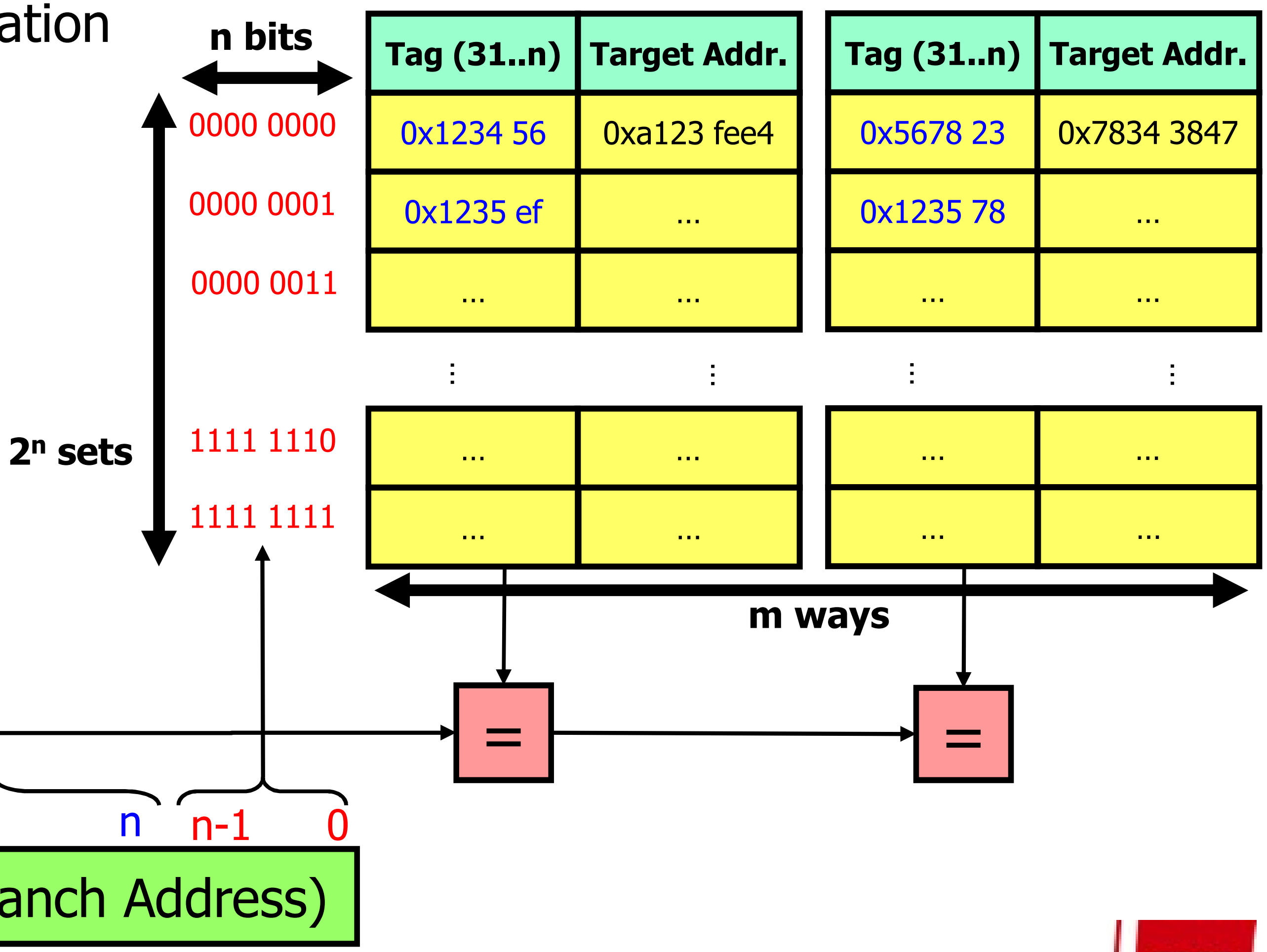
Target Prediction
Branch Target Buffers

Simplescalar configuration

BTB config

<#sets> <assoc.>

-bpred:btb 2^n m



Lab Outcomes

- Static prediction and software optimizations
- Dynamic prediction and aliasing
- Adaptive dynamic prediction basics
- Target prediction associativity

Benchmarks & Configurations

- C and assembly benchmarks are found in *test/src/*

  while.c, while.s, dowhile.c, dowhile.s, sqrt.c, strcpy.c

- Binary benchmarks are found in *test/bin.little/*

  sqrt, strcpy, test-math…

- Configuration files are found under *config/*

  a_nottaken.cfg, a_bimod.cfg, a_2lev.cfg, …

Static Prediction Benchmark

- Consider 2 versions of loop execution with static branch prediction: **branch not take**

```
#While
#define  N   10000
int main(void){
    long i;
    i=0;
    while (i<N){
        i++;
    }
    return (0);
}
```

```
#doWhile
#define  N   10000
int main(void){
    long i;
    i=0;
    do {
        i++;
    }
    while (i<N);
    return (0);
}
```

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Benchmark in Assembly

While

```
        sw      $0,16($fp)       } i=0;

$L2:    lw      $2,16($fp)
        slt     $3,$2,10000      } while (i<N)
        beq     $3,$0,$L3

        lw      $3,16($fp)
        addu    $2,$3,1          } i++;
        move    $3,$2
        sw      $3,16($fp)

        j       $L2

$L3:    move    $2,$0            } return(0);
```

doWhile

```
        sw      $0,16($fp)       } i=0;

$L2:    lw      $3,16($fp)
        addu    $2,$3,1          } i++;
        move    $3,$2
        sw      $3,16($fp)

        lw      $2,16($fp)
        slt     $3,$2,10000      } while (i<N)
        bne     $3,$0,$L2

        move    $2,$0            } return(0);
```

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

More on Assembly

- We use a simplescalar gcc compiler (ssgcc)
  - C → assembly → binary
- The compiler generates a startup code section in addition to the assembly provided
  - Stack pointer
  - Frame pointer
- The execution of the loops uses approximately 85% of the binary execution time
- The loop counter *i* is allocated a memory address and not a register

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Useful Commands

- Compile assembly

```
ssgcc  -g -o tests/<assembly test.s> -S tests/<test.c>
```

- Compile binary

```
ssgcc  -g -o tests/<binary test>  tests/< assembly test.s>
```

- Run a simulation

```
soo -config config/<filename>.cfg tests/<test> 2&> <output>.out
```

- Observe trace with pipeview

```
pv <filename>.trc | less
```

Jumps in Simplescalar

In simplescalar architecture jump instruction has to be predicted like a branch

Branch not taken will always fail in predicting a jump

Static Prediction

- Given the assembly on slide 12, how many times the static branch predictor, branch not taken, predicts correctly?
  - while has 1 branch and 1 jump, jump undergoes prediction
  - dowhile has 1 branch only

Static Prediction Simulation

- Verify your previous analysis by running a simulation
  - Generate a binary for each loop

```
ssgcc -g -o while  tests/src/while.s
ssgcc -g -o dowhile tests/src/dowhile.s
```

  - Launch a simulation for each loop, configuration a_nottaken.cfg

```
soo -config config/a_nottaken.cfg while 2&> while.out
soo -config config/a_nottaken.cfg dowhile 2&> dowhile.out
```

  - Search for “bpred lookups” in output file to find prediction stats

Optimized Assembly

- Generate an optimized assembly for the two loops by compiling with the **-O** optimization flag

```
ssgcc -g -O -o while_opt.s -S  tests/src/while.c

ssgcc -g -O -o dowhile_opt.s -S  tests/src/dowhile.c
```

- Compare the two assembly files
  - How many times the branch predictor predicts correctly now?

Optimized Binary

- Generate an optimized binary from the loops by compiling with the **-O** optimization flag

```
ssgcc -g -O -o while_opt  tests/src/while.c
ssgcc -g -O -o dowhile_opt  tests/src/dowhile.c
```

- Launch a simulation with the optimized binaries to verify your previous analysis

```
soo -config config/a_nottaken.cfg while_opt 2&> while_opt.out
soo -config config/a_nottaken.cfg dowhile_opt 2&> dowhile_opt.out
```

- If jumps were not predicted, how would static prediction perform?

Dynamic Prediction

- Launch a simulation with bimodal prediction for each loop
  - Use configuration a_bimodal.cfg

```
soo -config config/a_bimod.cfg while 2&> while_dynamic.out
soo -config config/a_bimod.cfg dowhile 2&> dowhile_dynamic.out
```

- Compare the prediction accuracy with static prediction

Bimodal Predictor

- Consider bimodal configuration with sqrt and strcpy
- Vary the size of the prediction table
  - "-bpred:bimod <table size>" for 2, 8, 32, 256, 4096
  - Uncomment the parameter in the config file
  - Note: sqrt requires an input argument (100 in this case)

```
soo -config config/a_bimod.cfg  tests/bin.little/sqrt  100 2&> sqrt.out
soo -config config/a_bimod.cfg  tests/bin.little/strcpy 2&> strcpy.out
```

- Show the effect on performance and explain the results

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Branch Target Buffer Associativity

- Consider the BTB associativity levels
- Vary BTB associativity and comment on the performance
  - Use test-math benchmark and configuration a_bimod.cfg
  - Compare IPC for core performance, Target and address prediction rate for prediction performance

```
soo -config config/a_bimod.cfg  tests/bin.little/test-math 2&>
btb.out
```

Look at the next slide

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Branch Target Buffer Associativity

	Sets	Assoc.	IPC	Addr-rate	Dir-rate
1	512	1			
2	256	2			
3	128	4			
4	64	8			
5	16	32			
6	1	512			

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Branch Target Buffer Performance

- Explain the performance results obtained in the table
  - Address prediction rate
  - IPC
  - Direction prediction rate

- What is the trade off between performance improvement and hardware cost of BTBs ?

2-level Adaptive Predictor

- The advantage of the two-level adaptive predictor is that it can quickly learn to predict a repetitive pattern

- Can you write a c-code that performs better with an adaptive over a bimodal predictor?
  - Hint: Come up with a simple pattern in your code