High Performance Computing Platforms

Step4 CA0603: Prediction and speculation

http://archlab.naist.jp/Lectures/ARCH/ca0603/ca0603e.pdf

Download one of following templates, fill in by handwriting, and

Send PDF (scan/photo)

To: naist.report@gmail.com

Subject: 4092-xxxxxxx (student ID)

http://archlab.naist.jp/Lectures/ARCH/ca0603/ca0603e.docx

These links are in http://archlab.naist.jp/Lectures

Prediction and Speculation for Instruction Stream

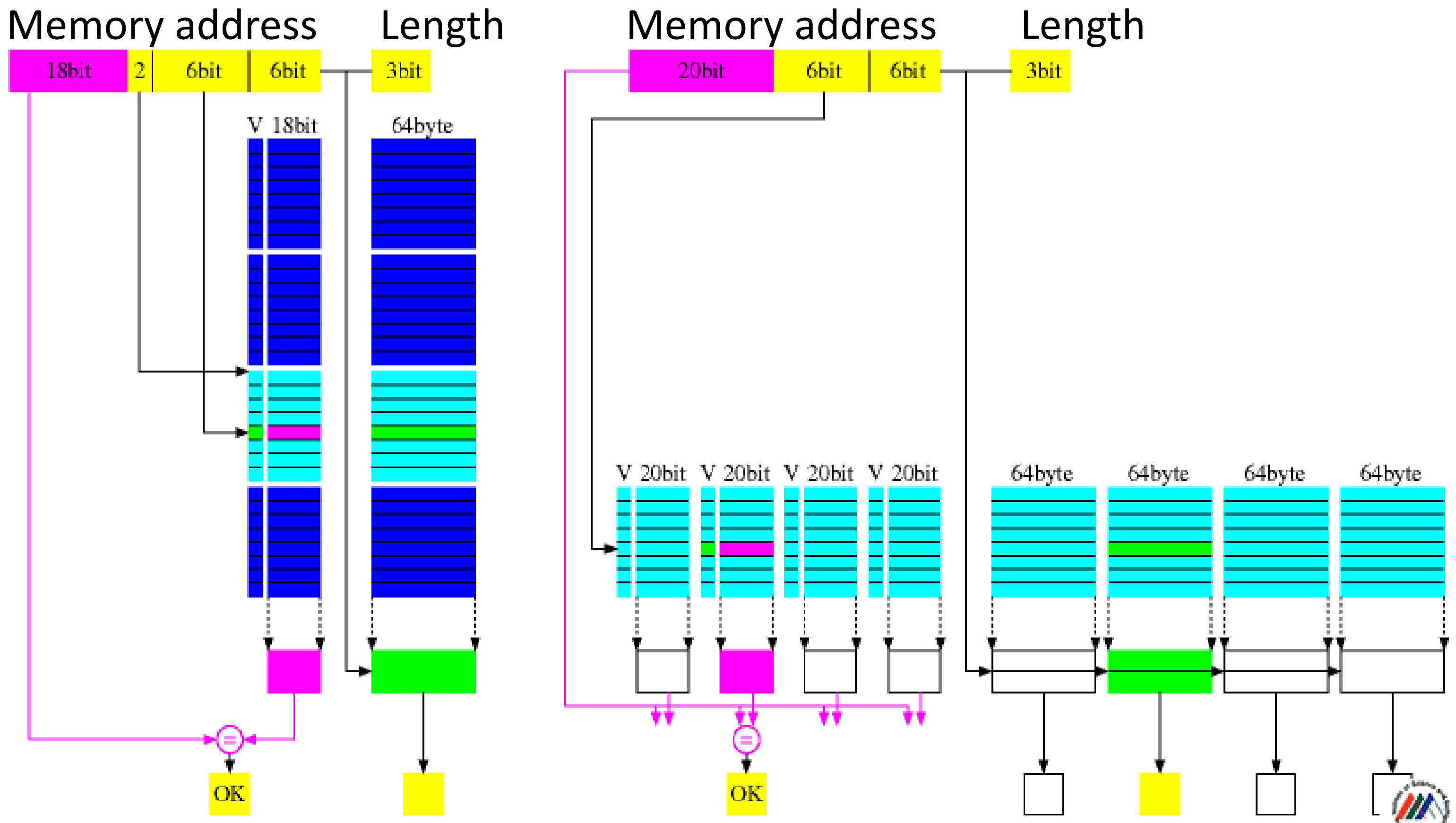
Instruction cache prefetch

Background

- Instruction supply speed is important to maximize ALU usage
- Even branch is exist, how to supply instruction from memory?

Topics

- Prefetch before cache miss is actually occurred
- Timing and Preciseness are important (Neither too early or too late)



Instruction cache prefetch (Various ideas)

Avoiding Next Line Prefetching

A.J.Smith: Cache memories, Computing Surveys, 14(3), Sep (1982)

- Next prefetch flag is added in every line
- When cache miss is occurred, set this frag in the line to ON
- When flag in next line is ON, do not prefetch and reset the flag to OFF

Target Prefetching

W.Hsu, et al.: A performance study of instruction cache prefetching methods, IEEE trans. on comp., 47(5), May (1998)

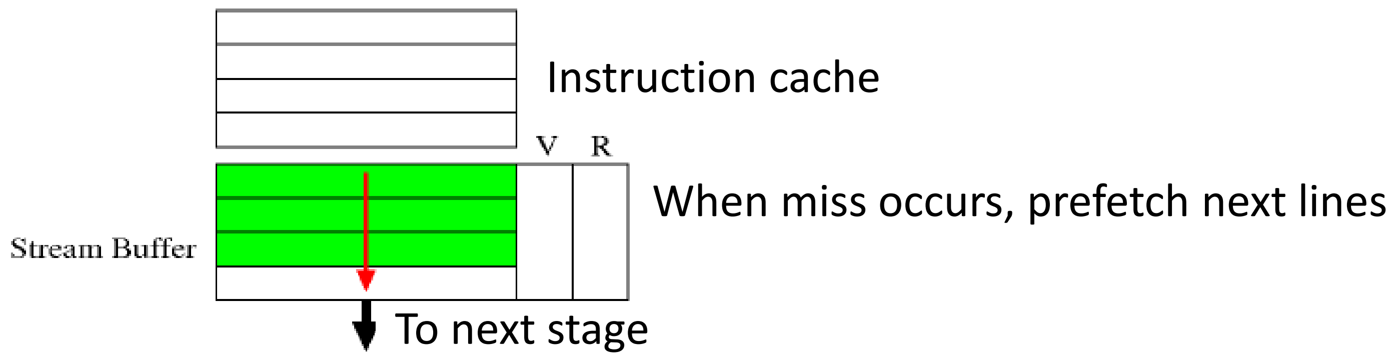
- Recode “next” line in additional table
- Too late decision for modern “too slow” main memory

More aggressive mechanisms are required

Instruction cache prefetch (Various ideas)

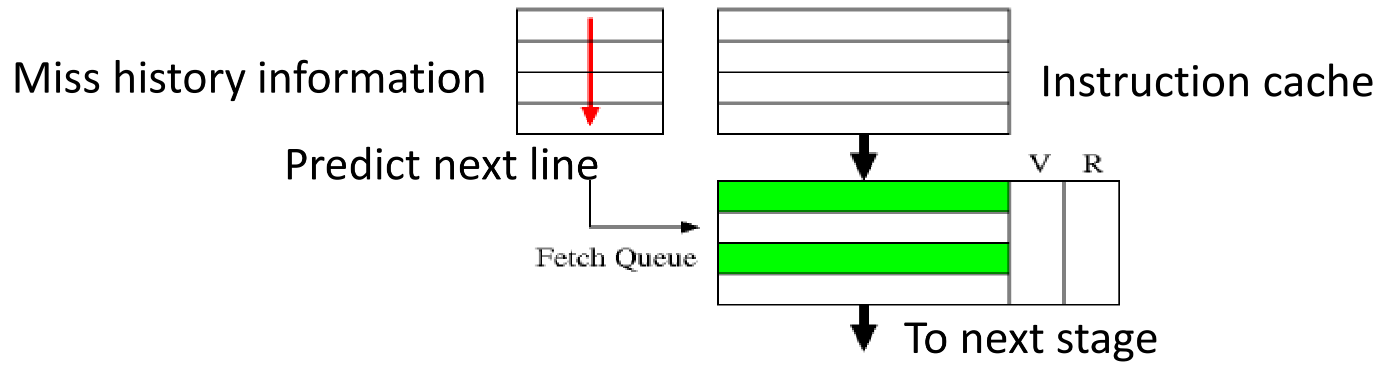
Stream Buffers

N.Jouppi: Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers, ISCA, May (1990)



Out-of-order Fetch

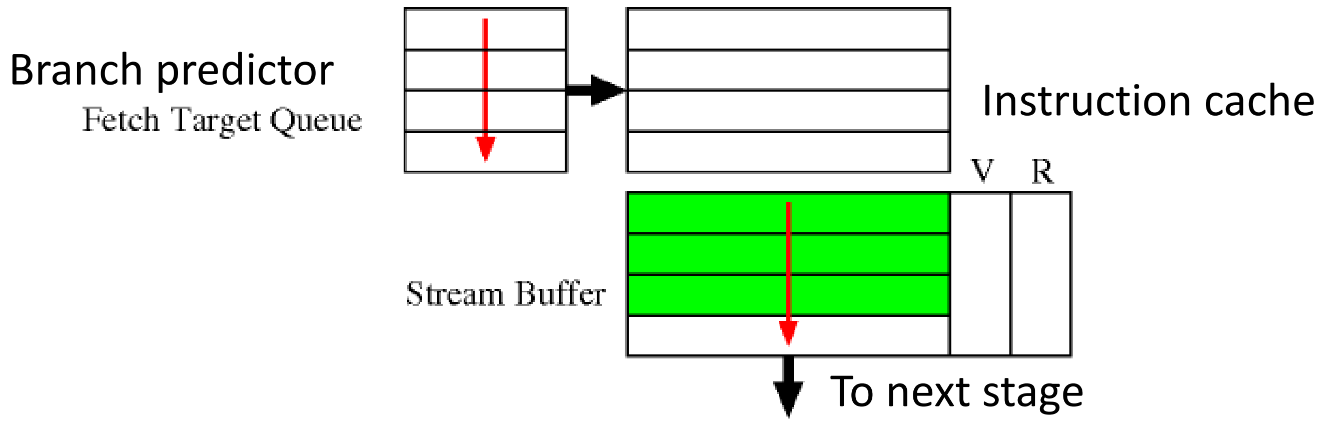
J.Stark, et al.: Reducing the performance impact of instruction cache misses by writing instructions into the reservation stations out-of-order, MICRO, Dec (1997)



Instruction cache prefetch (Various ideas)

Fetch Directed Prefetching

G.Reinman et al.: Fetch directed instruction prefetching, MICRO, Nov (1999)



Effective prefetch relies on precise branch prediction

- Actual branch prediction may make mistake
- Wrong prediction is not always useless
- If mispredict at beginning or end of loop, total performance may be
- improve
- If compiler minimize conditional branches, more effective

Some architecture (ex. IA64) has dedicated prefetch instruction

Branch prediction (Ideas of Branch direction prediction)

Static prediction

backward branches should be end of loop, then predict as taken

If instruction has “hint bit”, which set by compiler, just follow it

Dynamic prediction

Prediction

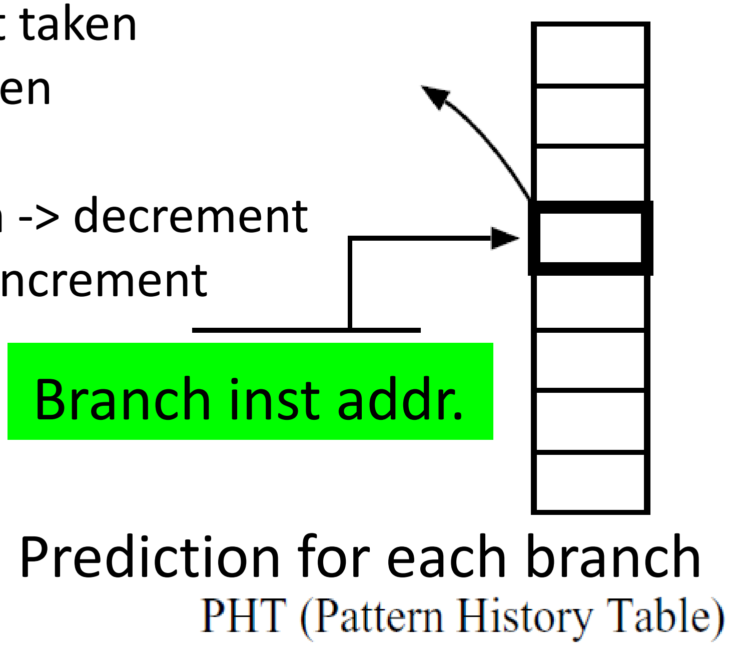
00,01 not taken

10,11 taken

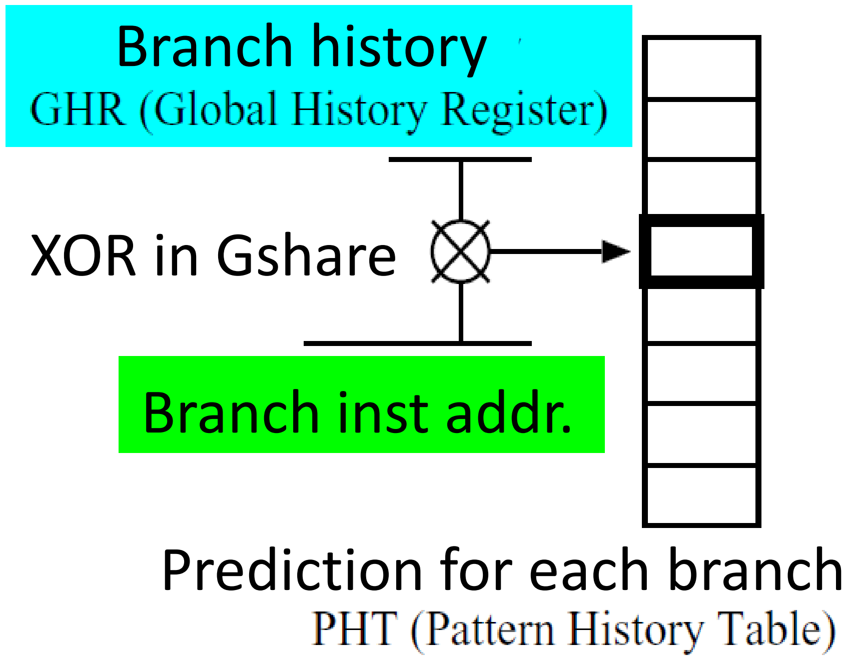
Update

not taken -> decrement

taken -> increment



(a) Record history for each branch instruction


Branch history
GHR (Global History Register)
XOR in Gshare
Branch inst addr.
Prediction for each branch
PHT (Pattern History Table)

(b) Record history for each branch sequence

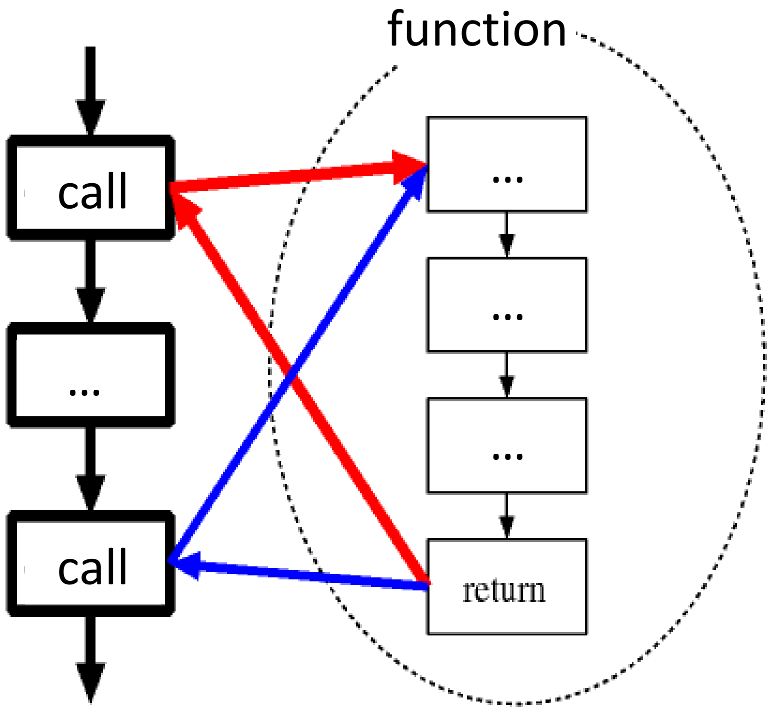
Branch prediction (How to predict return address)

Function call

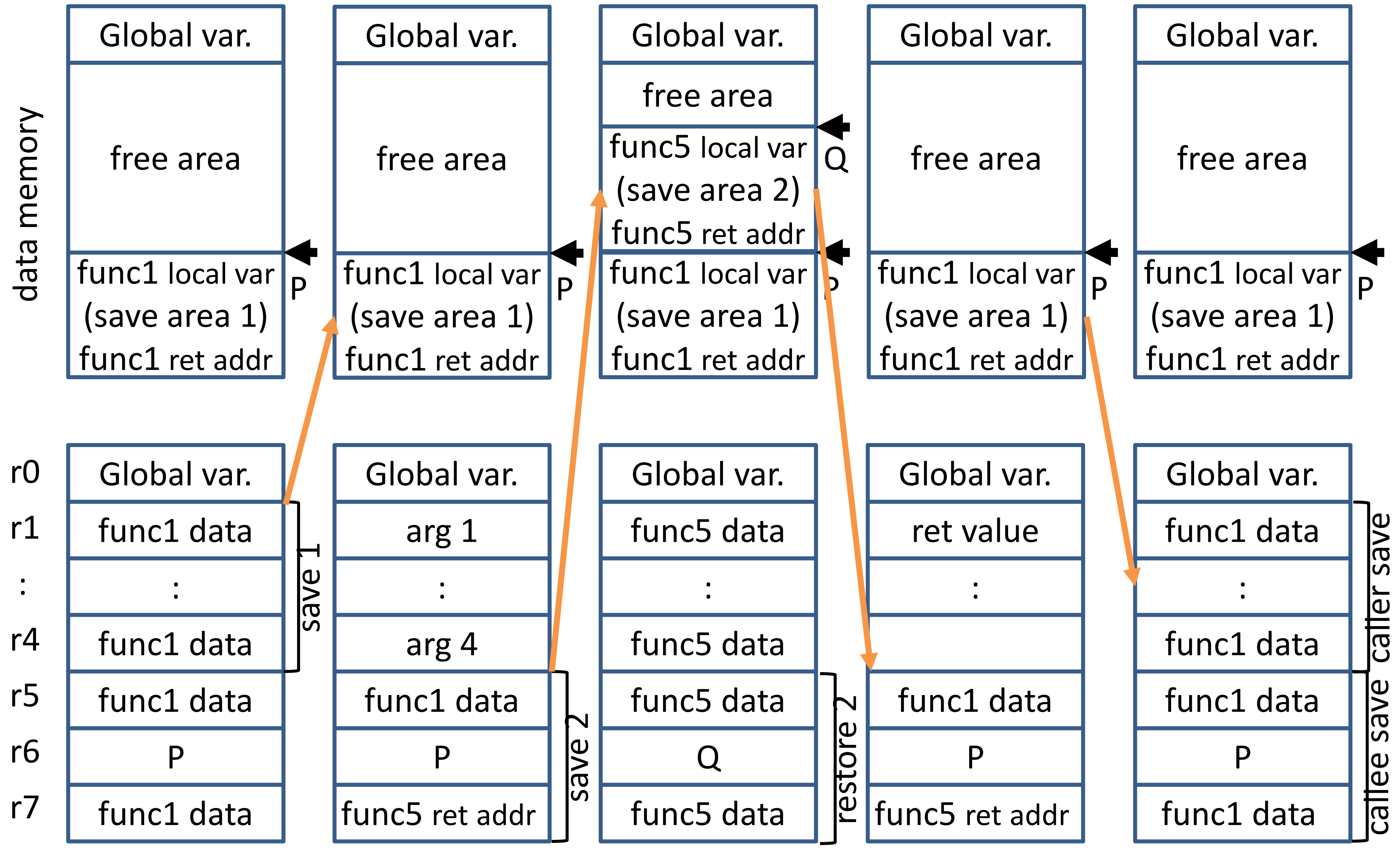
```
        sub();    /*call function */
        :
        return;  /* return to prev func */
```

Ex. ARM architecture

```
        bl sub;    /*save return addr in lr */
        :
    sub:
        :
        return;  /* return to lr */
```



Function call and stack frame



(a) func1 executing

(b) func5 just called

(c) func5 executing

(d) func5 returning

(e) func1 executing

Branch prediction (How to predict return address)

For PC relative branch, branch target is available at decode stage

For return, return address is saved in main memory, it is available
When load it from main memory.

- Due to long latency of main memory, special mechanism is necessary

Save return address in special register (Return Address Stack)

- When Call is decoded, push next address
- When Return is decoded, pop and use it as a predicted address
- When Return is executed, check correctness. If mismatch, then misprediction (need recovery)

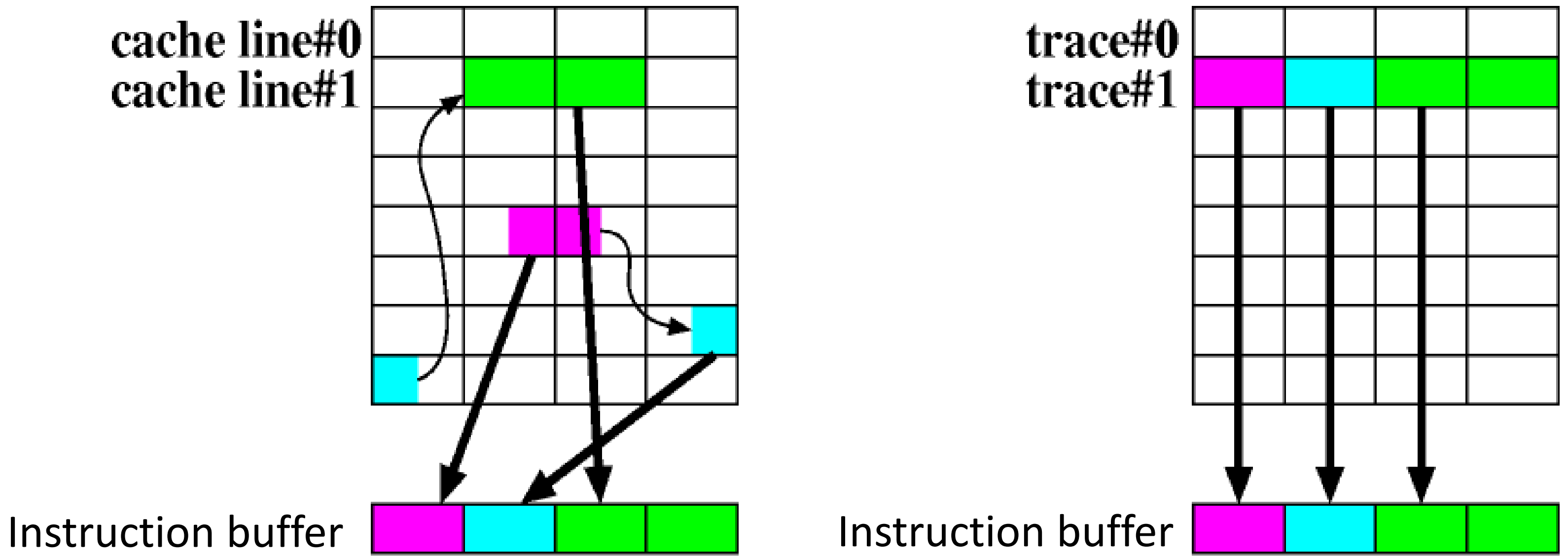
Trace cache

Background

- **Branch causes fragmentation of instruction cache**
- **When recording, can we defragment it?**

Topics

- **If sequence of basic block is same, we can reuse it**
- **What should we record?**

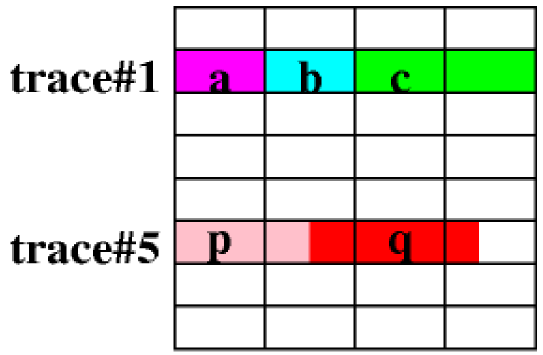


Trace cache

Identify entry with start address and branch pattern

- **Max number of basic blocks is limited by bit length of branch field**
- **Max number of instructions is limited by entry length of trace cache**

By including branch pattern, multiple entries that have same start address can be registered



start	branch	mask	next	target
a	101-	1110	d	p
p	1---	1000	r	x

Recode branch history in the trace

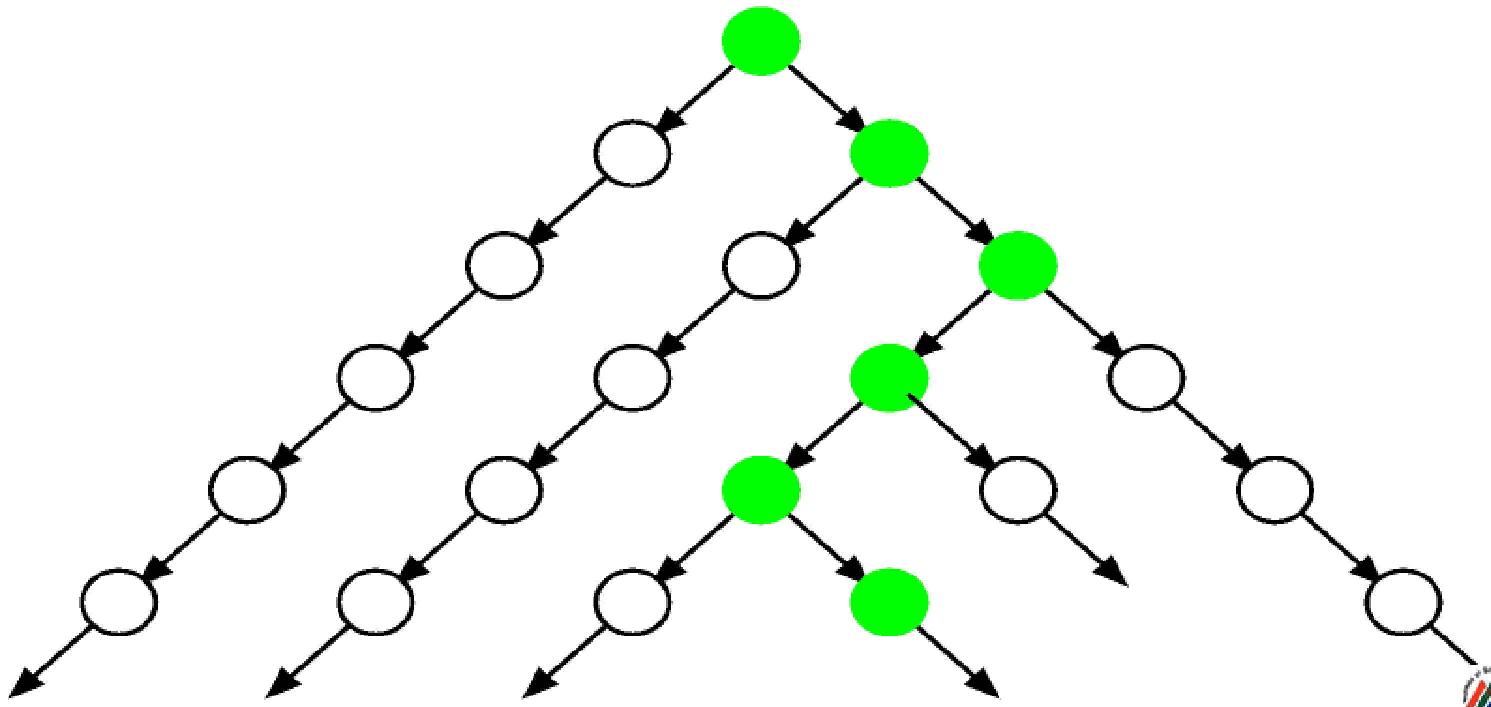
Multipath execution

Background

- **Why we predict one of taken or not taken?**
- **Why not execute BOTH?**

Topics

- **Execution is tree structure. Branch is node**
- **Which branch should we executed**



Multipath execution

Multipath execution is effective for non-predictable branch

Important for guaranteeing response time in embedded system

Use additional PC for prefetch. Each PC fetch different instruction stream simultaneously
Unlike thread speculation (explain later) , data is not predicted

- Extended super scalar
- Chip Multi Processor (CMP)

- See references for more detail

Prediction and Speculation for Data (Operand)

Data cache prefetch

Background

- **Same as instruction, data supply is important for good performance**
- **Some instructions are sequential, but limited data is sequential**
- **If programmer knows data location and reference order, software prefetching is effective**

Topics

- **How to prefetch data before cache miss**
- **Same as instruction, timing and preciseness is important**
  - **Prefetch only evident data is too pessimistic**
  - **Prefetch too much causes contention**
  - **Too early, evict before access**
  - **Too late, no effect**

Data cache prefetch (software approach)

Software prefetch

- Specific mechanism which can prefetch while cache miss is required
- Transfer data from memory to cache for specified address
- Some processors equip prefetch dedicated buffer

For array data

- If 2 arrays are used, one should be always in cache by loop transformation, the other is aided by prefetch

For indexed array data (index is given by other array)

- prefetch index array
- prefetch main array
- prefetch index array again (main array may evict index array)

Data cache prefetch (software approach)

Data structure with pointer

- **Idea 1 : prefetch all possible candidate**
  Cannot prefetch the data after the next
- **Idea 2 : relocate frequent access pattern to sequential**
  Hard to keep it sequential when add/delete some node
- **Idea 3 : Add prefetch pointer named Jump Pointer**
  Cannot prefetch first node
- **Idea 4 : Add dedicated pointer array for prefetch**
  Hard to maintain this array

Data cache prefetch (Hardware approach)

Hardware prefetch

- Predict future access by access pattern history

Stride/Sequential Prefetching

- Add dedicated buffer
- Prefetch sequential or fixed interval (stride) from main memory

Correlation Prefetching

- Create pattern history table
- Effective for indexed array or pointer chain (when patter is same)

Content-Based Prefetching

- If detect pointer value is loaded, prefetch it
- Check similarity of past access address
- Cannot prefetch array data

Address prediction

Background

- **If dependency between load/store instructions can be detected, further speed up is possible by reordering**
- **When load will be executed after write, effective addresses are different load instruction can be executed before store. Following instructions can be started**

Topics

- **Dependency between registers can be easily detected in decode stage**
- **Dependency between memory access addresses can be detected after address calculation**
- **How to predict address dependency precisely?**

Address prediction (Deterministic)

Compiler optimization

- Compiler re-order load/store instructions by static analysis

Pre-execution by hardware

- Add special hardware that pre-execute only address calculation related instructions

Pre-execution by helper thread

- Compiler generate additional instructions that only calculate memory address and pre-execute it

Address prediction (Speculative)

Without history table

- **Predict next address from previous access address**

With history table

- **Indexed by part of PC**

Last Value	**expect repeat access**
Stride Based	**expect same distance**
Context Based	**use global history**
Hybrid	**combination of several method**

Data speculation

Background

- If not just address but load data is predictable, even cash miss occurs following execution can be continued
- Not only exact value prediction but data dependency prediction is also part of data speculation

Topics

- Since answer of branch prediction is taken or not taken, even random prediction can correctly predict 50% of branches. Data prediction have to predict multiple bit
- Miss penalty of data misprediction is larger than just waiting

- Miss penalty may cause performance degradation
- No commercial processor has been released
- Let’s take a general view of research of data speculation

Data speculation (data prediction)

Same as address prediction, predict with history table

- Indexed by part of PC

Last Value	expect repeat value
Stride Based	expect same distance
Context Based	use global history
Hybrid	combination of several method

Miss penalty is huge, low precision predictions should be suppressed

- Precision should be predicted

Data speculation (dependency prediction)

For out-of-order execution, exact address is not required
Only address matching information is required
If effective address is not same, load and store can be reordered

- **If two addresses ware same in previous execution, there address will be same in next execution**
- **If two addresses are same, store data can be bypassed to following load instruction before memory access**
- **Dependency prediction is the most important topic for speculative multi threading (see reference)**

Summary

Speculative execution can be divided into …

- Instruction prefetch and branch prediction
- Memory address prediction
- Data prediction

Researcher proposes “promising” speculation

- In general, to get large performance overhead, hardware will be complicated and miss penalty will be large

Opportunity of hardware speculation is limited
Software support will be a key for further improvement

- Compiler must guarantee correct execution
  Aggressive speculation is not easy

That’s all for today