Constructive Computer Architecture:

Branch Prediction

Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

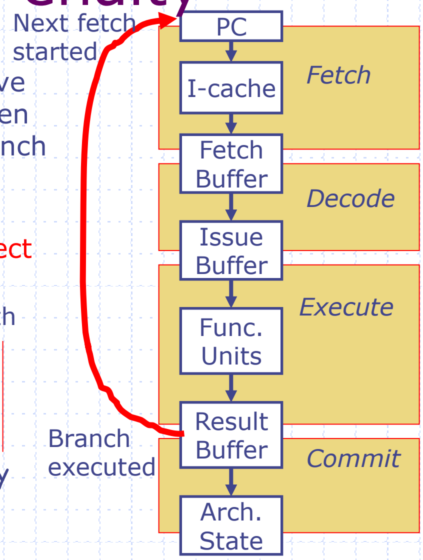
Contributors to the course material

- Arvind, Rishiyur S. Nikhil, Joel Emer, Muralidaran Vijayaraghavan
- Staff and students in 6.375 (Spring 2013), 6.S195 (Fall 2012), 6.S078 (Spring 2012)
  - Asif Khan, Richard Ruhler, Sang Woo Jun, Abhinav Agarwal, Myron King, Kermin Fleming, Ming Liu, Li-Shiuan Peh
- External
  - Prof Amey Karkare & students at IIT Kanpur
  - Prof Jihong Kim & students at Seoul Nation University
  - Prof Derek Chiou, University of Texas at Austin
  - Prof Yoav Etsion & students at Technion

Control Flow Penalty

- Modern processors may have > 10 pipeline stages between next PC calculation and branch resolution !
- How much work is lost if pipeline doesn't follow correct instruction flow?
  - Loop length x pipeline width
- What fraction of executed instructions are branch instructions?

superscalarity



How frequent are branches?

Blem et al [HPCA 2013]   Spec INT 2006 on ARM Cortex 7

ARM Cortex-A9; ARMv7 ISA

Benchmark	Total Instructions	branch %	load %	store %	other %
astar	1.47E+10	16.0	55.6	13.0	15.4
bzip2	2.41E+10	8.7	34.6	14.4	42.2
gcc	5.61E+09	10.2	19.1	11.2	59.5
gobmk	5.75E+10	10.7	25.4	7.2	56.8
hmmer	1.56E+10	5.1	41.8	18.1	35.0
h264	1.06E+11	5.5	30.4	10.4	53.6
libquantum	3.97E+08	11.5	8.1	11.7	68.7
omnetpp	2.67E+09	11.7	19.3	8.9	60.1
perlbench	2.69E+09	10.7	24.6	9.3	55.5
sjeng	1.34E+10	11.5	39.3	13.7	35.5
Average		8.2	31.9	10.9	49.0

Every 12^{th} instruction is a branch

How frequent are branches?

Blem et al [HPCA 2013]   Spec FP 2006 on ARM Cortex 7

	ARM Cortex-A9; ARMv7 ISA				
Benchmark	Total Instructions	branch %	load %	store %	other %
bwaves	3.84E+11	13.5	1.4	0.5	84.7
cactusADM	1.02E+10	0.5	51.4	17.9	30.1
leslie3D	4.92E+10	6.2	2.0	3.7	88.1
milc	1.38E+10	6.5	38.2	13.3	42.0
tonto	1.30E+10	10.0	40.5	14.1	35.4
Average		12.15	4.68	1.95	81.22

Every 8th instruction is a branch

How frequent are branches?

Blem et al [HPCA 2013]   Spec INT 2006 on X86

	core i7; x86 ISA				
Benchmark	Total Instructions	branch %	load %	store %	other %
astar	5.71E+10	6.9	19.5	6.9	66.7
bzip2	4.25E+10	11.1	31.2	11.8	45.9
hmmer	2.57E+10	5.3	30.5	9.4	54.8
gcc	6.29E+09	15.1	22.1	14.1	48.7
gobmk	8.93E+10	12.1	21.7	13.4	52.7
h264	1.09E+11	7.1	46.8	18.5	27.6
libquantum	4.18E+08	13.2	39.3	6.8	40.7
omnetpp	2.55E+09	16.4	28.6	21.4	33.7
perlbench	2.91E+09	17.3	25.9	16.0	40.8
sjeng	2.11E+10	14.8	22.8	11.0	51.4
Average		9.4	31.0	13.4	46.2

Every 10th or 11th instruction is a branch

How frequent are branches?

Blem et al [HPCA 2013] Spec FP 2006 on X86

		core i7; x86 ISA			
Benchmark	Total Instructions	branch %	load %	store %	other %
bwaves	3.41E+10	3.2	51.4	16.8	28.7
cactusADM	1.05E+10	0.4	55.3	18.6	25.8
leslie3D	6.25E+10	4.9	35.3	12.8	46.9
milc	3.29E+10	2.2	32.2	13.8	51.8
tonto	4.88E+09	7.1	27.2	12.4	53.3
Average		3.6	39.6	14.4	42.4

Every 27^{th} instruction is a branch

Observations

- No pipelined processor can work without a next address prediction
- Control transfer happens every 8^{th} to 30^{th} instruction

Simplest Next Address Prediction (NAP)

- What is it?

  pc, pc+4, pc+8, …i.e., Jumps and Branch are predicted not taken

- Is this a good idea?

  yes, because most instructions are not control transfer instructions (reported 70% accuracy)

- Can we do better?

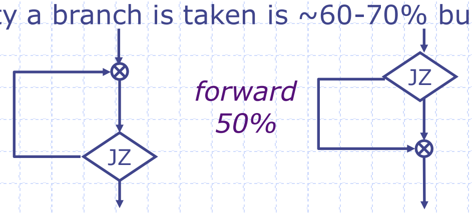
  Yes, by knowing something about the program or by learning from the past behavior of the program, aka dynamic branch prediction

Static Branch Prediction

Overall probability a branch is taken is ~60-70% but:

backward *90%*

forward *50%*

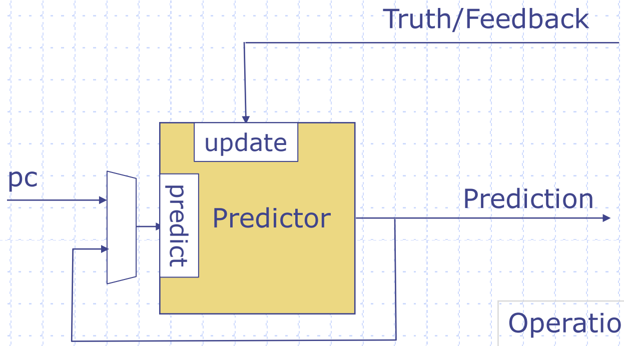


- ISA can attach preferred direction semantics to branches, e.g., Motorola MC88110
  - bne0 *(preferred taken)* beq0 *(not taken)*

- ISA can allow arbitrary choice of statically predicted direction, e.g., HP PA-RISC, Intel IA-64
  - reported as ~80% accurate

… but our ISA is fixed!

Dynamic Branch Prediction



Operations
- Predict
- Update

Dynamic Branch Prediction

learning based on past behavior

- Temporal correlation
  - The way a branch resolves may be a good predictor of the way it will resolve at the next execution
- Spatial correlation
  - Several branches may resolve in a highly correlated manner (a preferred path of execution)

Observations

- There is a plethora of branch prediction schemes – their importance grows with the depth of processor pipeline
- Processors often use more than one prediction scheme
- It is usually easy to understand the data structures required to implement a particular scheme
- It takes considerably more effort to
  - Integrate a prediction scheme in the pipeline
  - Understand the interactions between various schemes
  - Understand the performance implications

we will start with the basics ...

MIPS Branches and Jumps

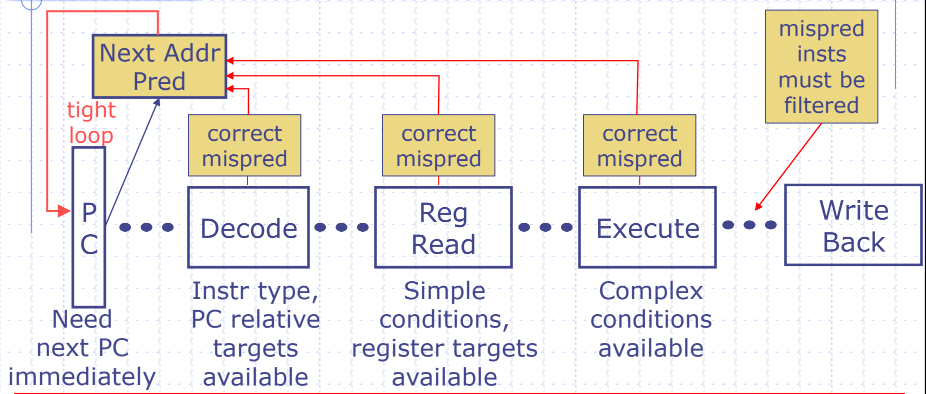
Each instruction fetch depends on some information from the preceding instruction:

1. Is the preceding instruction a taken branch?
2. If so, what is the target address?

Instruction	*Direction known after*	*Target known after*
J	After Inst. Decode	After Inst. Decode
JR	After Inst. Decode	After Reg. Fetch
BEQZ/BNEZ	After Exec	After Inst. Decode

A predictor can redirect the pc only after the relevant information required by the predictor is available

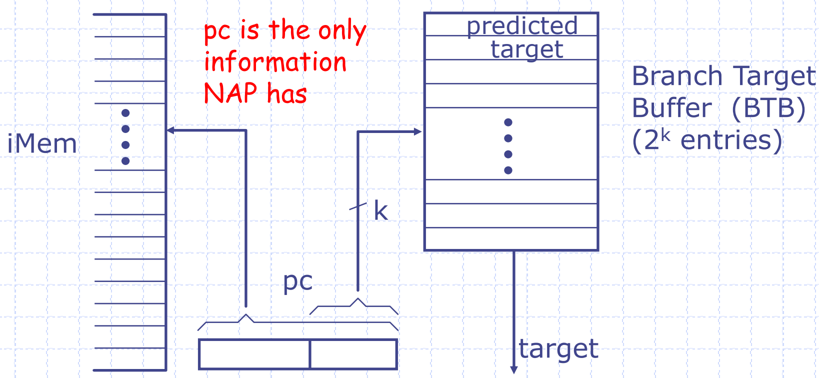
Overview of control prediction



Given (pc, ppc), a misprediction can be corrected (used to redirect the pc) as soon as it is detected. In fact, pc can be redirected as soon as we have a "better" prediction. However, for forward progress it is important that a correct pc should never be redirected.

Next Address Predictor (NAP)

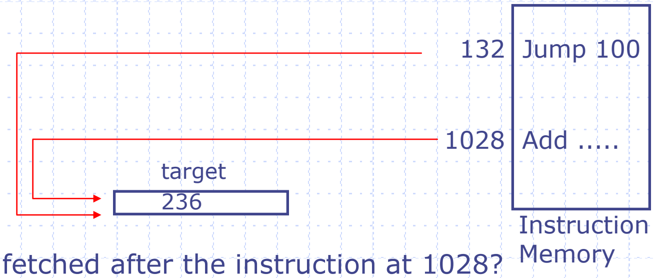
first attempt



Fetch: ppc = look up the target in BTB
Later check prediction, if wrong then kill the instruction and update BTB

Address Collisions

Assume a
128-entry
BTB



What will be fetched after the instruction at 1028?

NAP prediction = 236
Correct target = 1032

⇒ *kill* PC=236 and *fetch* PC=1032

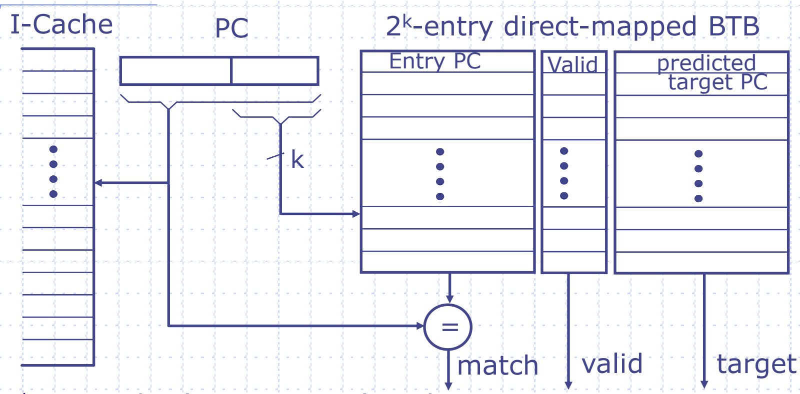
Is this a common occurrence? yes
Can we avoid these bubbles? yes

Use BTB for Control Instructions only

- BTB contains useful information for branch and jump instructions only
  ⇒ Do not update it for other instructions
- For all other instructions the next PC is (PC)+4!

How to achieve this effect without decoding the instruction?

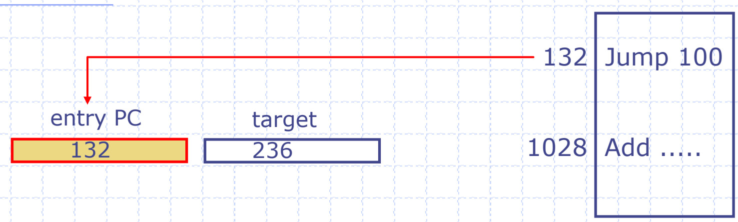
Branch Target Buffer (BTB)



- Keep the (pc, target pc) in the BTB
- pc+4 is predicted if no pc match is found
- BTB is updated only for branches and jumps

Permits ppc to be determined *before* instruction is decoded

Consulting BTB Before Decoding



- If the match for pc fails, pc+4 is fetched
  - pc=132, match succeeds, instruction at 236 is fetched
  - pc=1028, match fails, instruction at 1028+4 is fetched

    ⇒ *eliminates false predictions after ALU instructions*
- BTB contains entries only for control transfer instructions

  ⇒ *more room to store branch targets*

Even very small BTBs are very effective

Next Addr Predictor interface

```
interface AddrPred;
  method Addr predPc(Addr pc);
  method Action update(Redirect rd);
endinterface
```

Two implementations:

a) Simple PC+4 predictor
b) Predictor using BTB

Simple PC+4 predictor

```
module mkPcPlus4(AddrPred);
  method Addr predPc(Addr pc);
    return pc + 4;
  endmethod

  method Action update(Redirect rd);
  endmethod
endmodule
```

BTB predictor

```
module mkBtb(AddrPred);
  RegFile#(BtbIndex, Addr) ppcArr <- mkRegFileFull;
  RegFile#(BtbIndex, BtbTag) entryPcArr <- mkRegFileFull;
  Vector#(BtbEntries, Reg#(Bool))
       validArr <- replicateM(mkReg(False));
  function BtbIndex getIndex(Addr pc)=truncate(pc>>2);
  function BtbTag getTag(Addr pc) = truncateLSB(pc);
  method Addr predPc(Addr pc);
    BtbIndex index = getIndex(pc);
    BtbTag tag = getTag(pc);
    if(validArr[index] && tag == entryPcArr.sub(index))
      return ppcArr.sub(index);
    else return (pc + 4);
  endmethod
  method Action update(Redirect redirect); ...
endmodule
```


BTB predictor update method

Input redirect contains pc, the correct next pc and whether the branch was taken or not (to avoid making entries for not-taken branches

```
method Action update(Redirect redirect);
  if(redirect.taken)
   begin
    let index = getIndex(redirect.pc);
    let tag = getTag(redirect.pc);
    validArr[index] <= True;
    entryPcArr.upd(index, tag);
    ppcArr.upd(index, redirect.nextPc);
   end
endmethod
```

Integrating BTB in the 2-Stage pipeline

```
module mkProc(Proc);
  Reg#(Addr)           pc <- mkRegU;
  RFile                rf <- mkRFile;
  IMemory            iMem <- mkIMemory;
  DMemory            dMem <- mkDMemory;
  Fifo#(Decode2Execute) d2e <- mkFifo;
  Reg#(Bool)       fEpoch <- mkReg(False);
  Reg#(Bool)       eEpoch <- mkReg(False);
  Fifo#(Addr)    redirect <- mkFifo;
  AddrPred            btb <- mkBtb

  Scoreboard#(1) sb <- mkScoreboard;
  rule doFetch …
  rule doExecute …
```


2-Stage-DH pipeline
doFetch rule

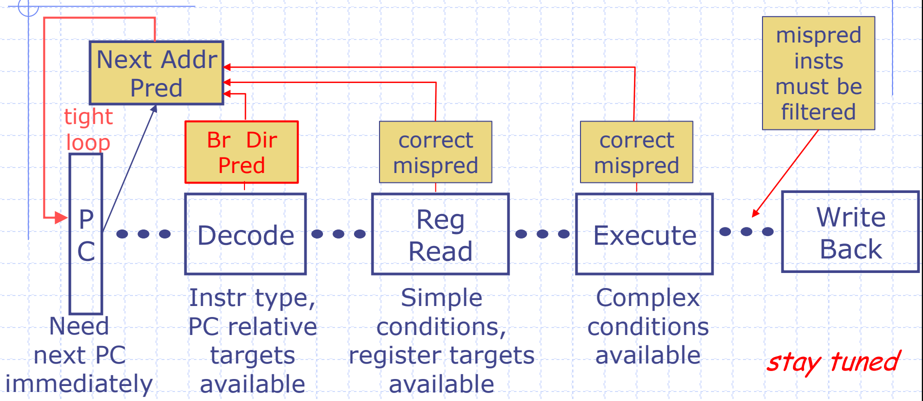
```
rule doFetch;
    let inst = iMem.req(pc);
    if(redirect.notEmpty) begin
      fEpoch <= !fEpoch;  pc <= redirect.first;
      btb.update(redirect.first); redirect.deq; end
    if(redirect.notEmpty && redirect.first.mispredict)
      begin pc <= redirect.first.ppc; fEpoch <= !fEpoch; end
    else begin
      let ppc = btb.predPc(pc)         ; let dInst = decode(inst);
      let stall = sb.search1(dInst.src1)|| sb.search2(dInst.src2);
      if(!stall)                       begin
         let rVal1 = rf.rd1(validRegValue(dInst.src1));
         let rVal2 = rf.rd2(validRegValue(dInst.src2));
         d2e.enq(Decode2Execute{pc: pc, nextPC: ppc,
                dIinst: dInst, epoch: fEpoch,
                rVal1: rVal1, rVal2: rVal2});
         sb.insert(dInst.rDst); pc <= ppc; end
          end
    endrule
```

2-Stage-DH pipeline doExecute rule

```
rule doExecute;
    let x = d2e.first;
    let dInst = x.dInst; let pc    = x.pc;
    let ppc   = x.ppc;   let epoch = x.epoch;
    let rVal1 = x.rVal1; let rVal2 = x.rVal2;
    if(epoch == eEpoch) begin
      let eInst = exec(dInst, rVal1, rVal2, pc, ppc);
      if(eInst.iType == Ld) eInst.data <-
        dMem.req(MemReq{op:Ld, addr:eInst.addr, data:?});
      else if (eInst.iType == St) let d <-
        dMem.req(MemReq{op:St, addr:eInst.addr, data:eInst.data});
      if (isValid(eInst.dst))
        rf.wr(validRegValue(eInst.dst), eInst.data);
      if(eInst.iType == J || eInst.iType == Jr || eInst.iType == Br)
        redirect.enq(Redirect{pc: pc, nextPc: eInst.addr,
            taken: eInst.brTaken, mispredict: eInst.mispredict,
            brType: eInst.iType, });
      if(eInst.mispredict) eEpoch <= !eEpoch;
    d2e.deq; sb.remove;
endrule
```

October 21, 2013 http://csg.csail.mit.edu/6.S195 L14-27

Multiple Predictors: BTB + Branch Direction Predictors



- Suppose we maintain a table of how a particular Br has resolved before. At the decode stage we can consult this table to check if the incoming (pc, ppc) pair matches our prediction. If not redirect the pc

October 21, 2013 http://csg.csail.mit.edu/6.S195 L14-28