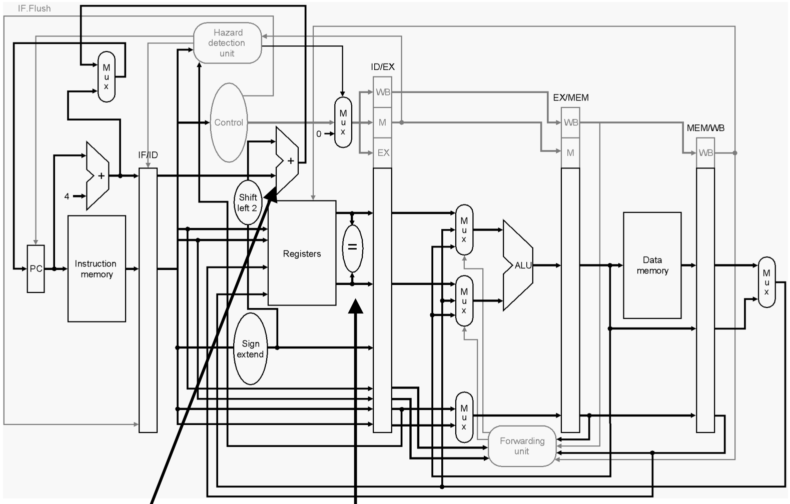
Detecting and Handling Exceptions and Interrupts

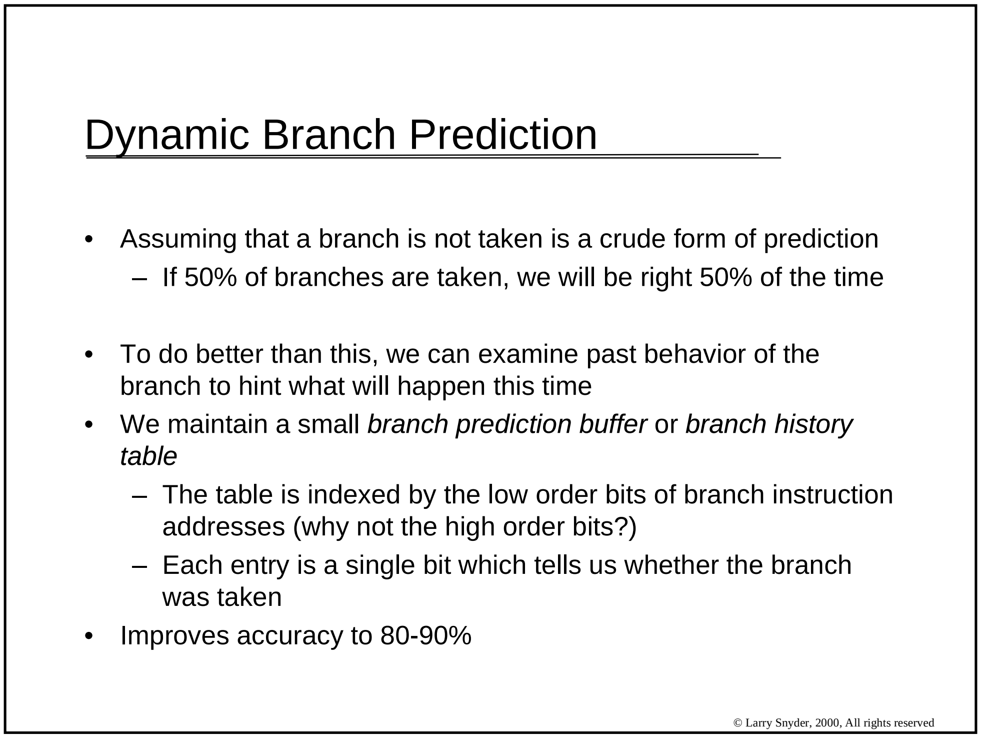
The datapath has to be prepared to handle unusual situations -- it cannot stop, but must keep going and recover

Branch Hazard Detection Logic



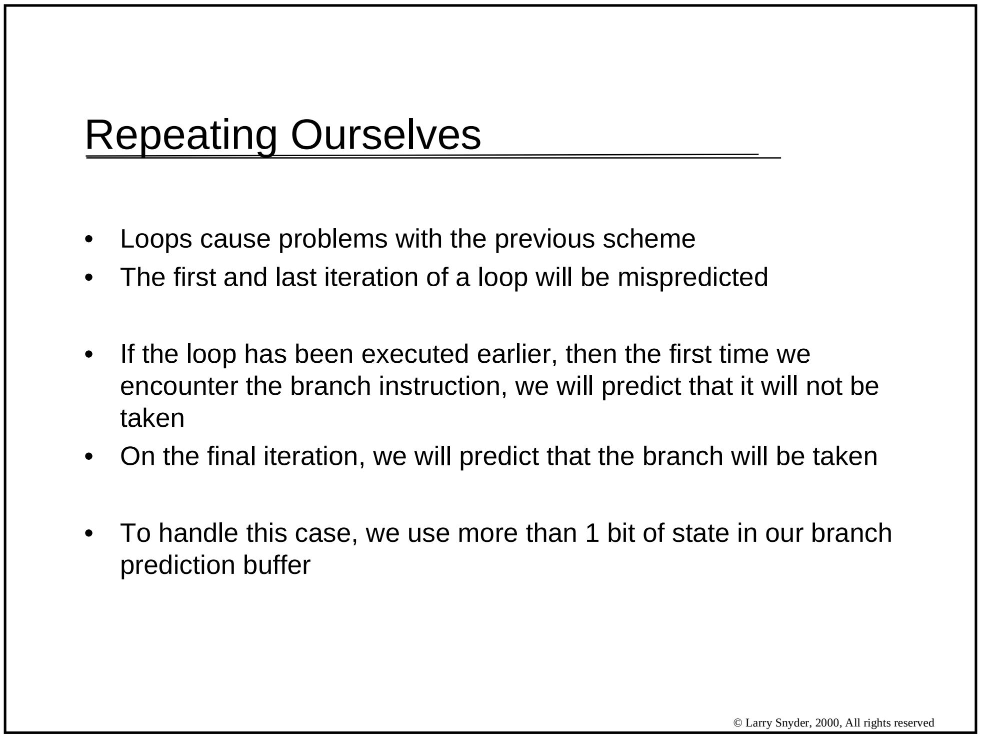
EA Computation Moved to ID

Fast Equality Test



Dynamic Branch Prediction

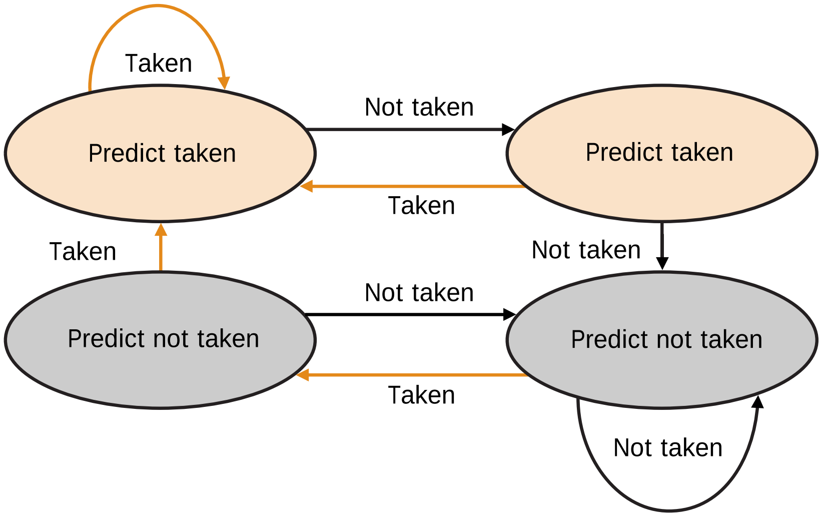
- Assuming that a branch is not taken is a crude form of prediction
  - If 50% of branches are taken, we will be right 50% of the time
- To do better than this, we can examine past behavior of the branch to hint what will happen this time
- We maintain a small *branch prediction buffer* or *branch history table*
  - The table is indexed by the low order bits of branch instruction addresses (why not the high order bits?)
  - Each entry is a single bit which tells us whether the branch was taken
- Improves accuracy to 80-90%



Repeating Ourselves

- Loops cause problems with the previous scheme
- The first and last iteration of a loop will be mispredicted
- If the loop has been executed earlier, then the first time we encounter the branch instruction, we will predict that it will not be taken
- On the final iteration, we will predict that the branch will be taken
- To handle this case, we use more than 1 bit of state in our branch prediction buffer

2-bit Prediction Scheme



Taken

Not taken

Predict taken

Predict taken

Taken

Taken

Not taken

Not taken

Predict not taken

Predict not taken

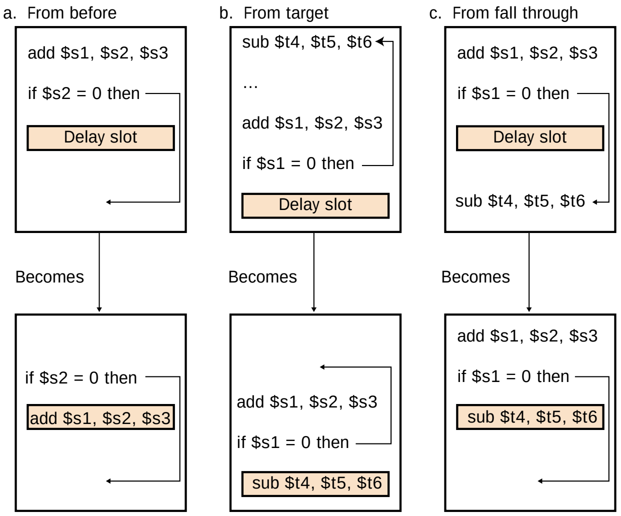
Taken

Not taken

Delayed Branches

- Make the control hazard an architectural feature
- The instruction following a branch is always executed
- The compiler or assembler has to find an instruction to fill this slot
  - If none can be found, a NOP has to be inserted
- The instructions scheduled into the delay slot must
  - EITHER always be executed whether the branch is taken or not
  - OR have no side-effects
- Less popular now since longer pipelines and multiple instruction issue mean the single delay slot does not help as much
- Dynamic predictors have increased in popularity as transistor density has increased

Playing the Slots



a. From before

add $s1, $s2, $s3

if $s2 = 0 then

Delay slot

Becomes

if $s2 = 0 then

add $s1, $s2, $s3

b. From target

sub $t4, $t5, $t6

…

add $s1, $s2, $s3

if $s1 = 0 then

Delay slot

Becomes

add $s1, $s2, $s3

if $s1 = 0 then

sub $t4, $t5, $t6

c. From fall through

add $s1, $s2, $s3

if $s1 = 0 then

Delay slot

sub $t4, $t5, $t6

Becomes

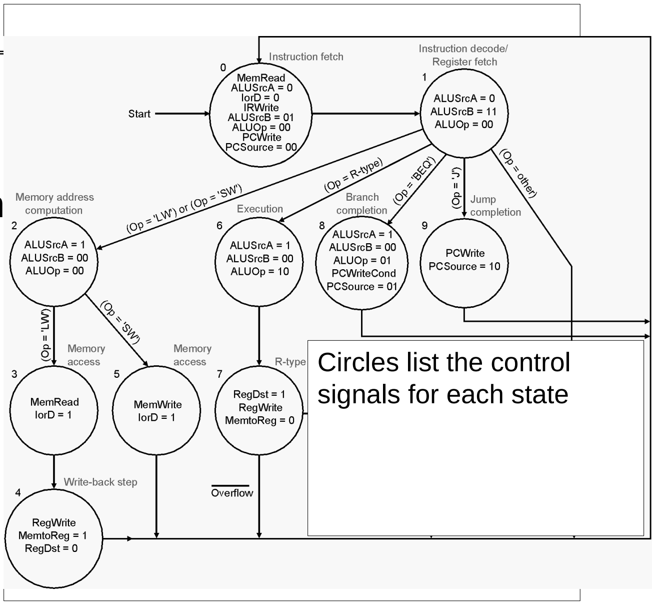
add $s1, $s2, $s3

if $s1 = 0 then

sub $t4, $t5, $t6

Abstracting Control

Finite state machine representation of datapath



Circles list the control signals for each state

Unusual Conditions Arise ...

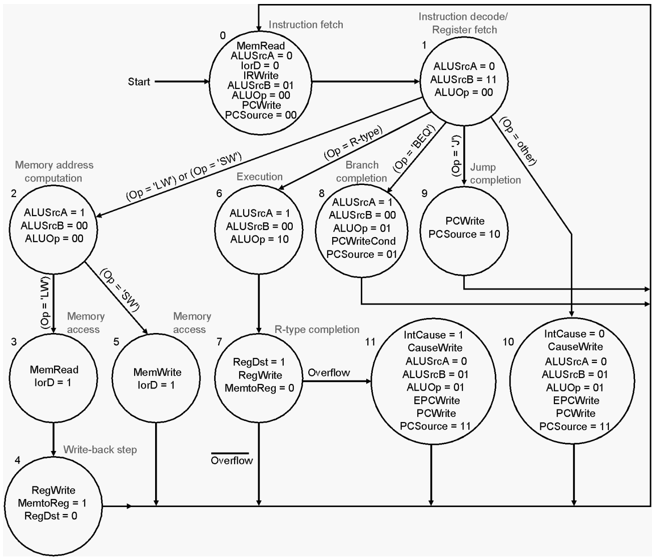
Classify the unusual things that can happen

- Exceptions -- unusual events that affect the datapath, regardless of whether they are internally or externally generated
- Interrupts -- externally generated events

Examples …

- I/O device requests
- Invoke operating system from user program
- Arithmetic overflow
- Undefined instruction
- Errors

Detection



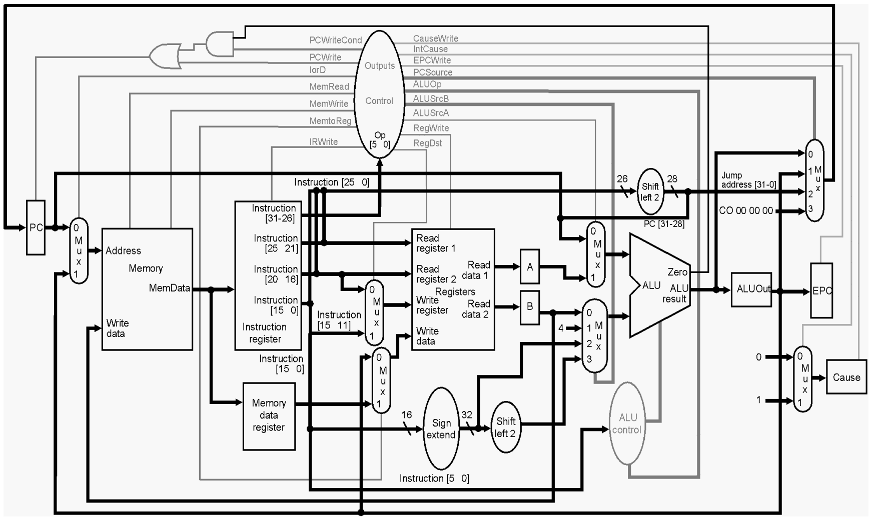
Exceptions in the Pipeline

- Handling exceptional conditions in the pipeline is difficult … multiple instructions are in process, but execution must be stopped at a coherent point
- The operating system will handle exceptions
- Two requirements …
    - Save the address of offending instruction in EPC, exception program counter
    - Transfer control to operating system

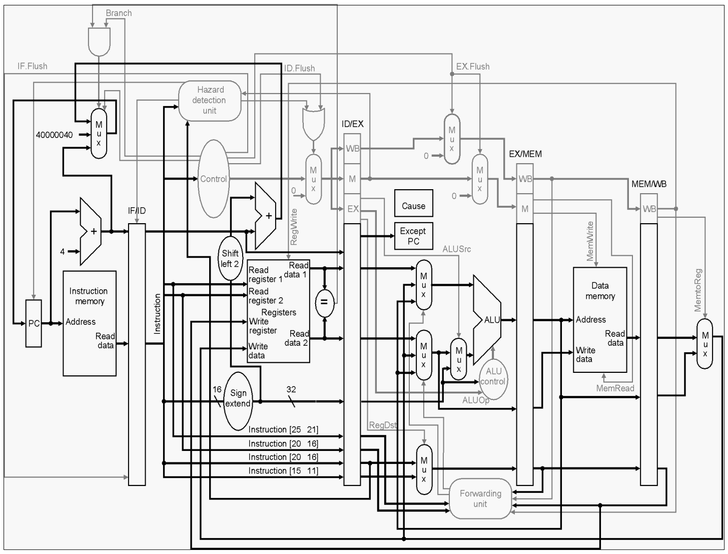
Keeping Track of What Happened

- The machine must tell the OS what happened
- A “cause register” or “exception flags” are bit sequences that the processor sets indicating errors
- “Vectored interrupts” allow the processor to jump to different locations in the operating system depending on the exceptional condition

Add-ins for Exceptions



Controls


Branch
IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
40000040
M u x
Control
ID/EX
WB
M
EX
EX/MEM
MEM/WB
Cause
Except PC
0
IF/ID
+
4
Shift left 2
RegWrite
Instruction memory
Address
Read data
PC
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
=
ALUSrc
ALU
ALU control
ALUOp
MemWrite
Data memory
Address
Read data
Write data
MemRead
MemtoReg
16
Sign extend
32
Instruction [25 21]
Instruction [20 16]
Instruction [20 16]
Instruction [15 11]
RegDst
Forwarding unit