Dynamic Issue & HW Speculation

Today's topics:

Superscalar pipelines

Dynamic Issue

Scoreboarding: control centric approach

Tomasulo: data centric approach

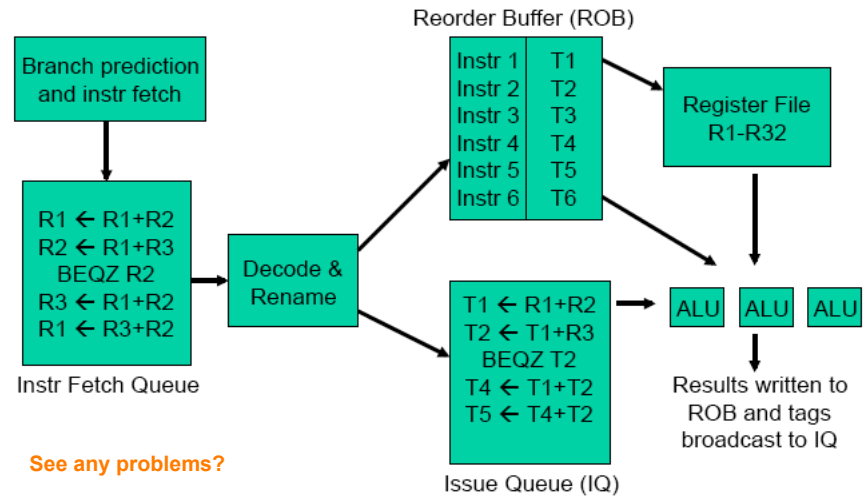
Raising the IPC Ceiling

- w/ single-issue IPC_{max} = 1
  - schedule as hard as you want and it's still the asymptote
    - » keeping things in order → lots of stalls
      - XU's finish out of order anyway
    - » when the transistor budget is high enough
      - just go with multiple issue
        - >= 4 issue common today ::= superscalar machines
- Superscalar issues: $issue_{width}$ = n
  - need n way capability in all pipeline stages
    - » fetch n – no worries fetch cache line of instructions/cycle
    - » decode n
      - get register values – problems?
    - » execute n
      - problems?
    - » mem n
      - problems? w/out of order completion?
    - » WB n
      - problems w/ out of order completion?

Fix OOO Completion Problem First

- **Enter the ROB (re-order buffer)**
  - **basic idea for now**
    - » **issue instructions in-order**
    - » **retire/commit instructions in order**
    - » **use an intermediate buffer to hold results**
      - **since destructive action to register file or memory must happen in order**
- **Other ROB niceties**
  - **helps w/**
    - » **speculation**
    - » **nullification**
    - » **exceptions**
  - **but first a simple example**

Reorder Buffer In Action



Several Issues

- WB stage is now the commit stage
  - ROB values move to the register file
    - » whoops if tags are in the issue queue
      - those values need to be renamed to the register name
      - seems complex – can you thing of a better way?

Several Issues

- WB stage is now the commit stage
  - ROB values move to the register file
    - » whoops if tags are in the issue queue
      - those values need to be renamed to the register name
      - seems complex – can you think of a better way?
  - IQ contains both register and tag fields
    - » w/ 1 bit to select which is valid
      - initially tag is selected
      - when tag is retired
        - broadcast to IQ and invert selector on a match
      - what about tag values in the pipe
        - only need to worry about entry into EX stage
        - compares needed there as well
        - ROB is WB stage so that's not a problem
        - MEM isn't a problem either WHY?
- Key observation
  - all destructive operations are done by the ROB commit /retire

Nullification & Exceptions

- If an exception happens
  - exception type is written to the ROB field
    - » note that one instruction could generate an exception in multiple stages
      - only care about the first one so no overwrite is allowed
- If some instruction is speculative
  - then predicate is written to the ROB field
  - note: predicate covers branch delay slots and effectively supports nullification
- WB stage in reality
  - try to retire n instructions per cycle
    - » if none have pending predicates or exceptions then retire
    - » in order retire → 1st member of n-instruction bundle w/ problem
      - retire the instructions before
      - nullify whatever is next in the bundle
      - take the exception and hold the rest

Decode Complexity

- ROB complicates ID significantly
  - operand fetch now has two sources
    - » register file or ROB field
      - hence an additional mux is required
  - rename takes some time
    - » structural issue requirements will help mitigate the performance penalty
- Bottom line
  - ID will no longer be a single cycle stage
- For register poor ISA's like x86
  - ROB slots effectively provides a renamed register pool
    - » actually it's not the right choice
      - Why?
      - remember the front-end back-end x86 thing

ROB Hazard Removal

- RAW
  - nothing changes here
    - » no way you can use a value before it's computed
    - » unless the value is predicted and predicated
      - only *some* academic papers think this is a reasonable idea
    - » hence instruction scheduling is required
- Wax
  - ROB renaming effectively removes this problem
    - » as long as enough ROB slots exist
    - » if not
      - then the instruction can't be issued and a NOP is injected in the pipe
- Note
  - stalling pipelines @ GHz frequencies is a problem
    - » hence NOPs are dynamically generated and pushed through the pipe
    - » any issues here?

EX Stages XU's

- Typical separation of XU's
  - ALU (int +/-, shift, logical (AND, OR, XOR, NOT)
  - int-multiply
  - int-divide
  - FP ops can be 32 or 64-bit (typically implement 64-bit)
    - » FP-add-sub
    - » FP-multiply
    - » FP-divide or FP-invert (1/x)
    - » FP-sqrt or FP-isqrt?
- Overlaps
  - Branch and Mem ops can be handled with an ALU
  - int mul or div can be handled by the FP equivalent
    - » a common choice is to have a int-mul but not an int-div
      - why?
  - actual choice influences structural issue rules

Structural Issue Rules

- Clearly vary by machine
- Example for a 6 issue machine
  - 2 ALU
  - 1 Branch
  - 1 Int Mul or Divide
  - 1 FP Add or Sub
  - 1 Mem
- Why does this make sense?
  - e.g. justification

Structural Issue Rules

- Clearly vary by machine
- Example for a 6 issue machine
  - 2 ALU or 1 ALU and 1 Int-Mul
  - 1 Branch
  - 1 FP Mul or Divide
  - 1 FP Add or Sub
  - 1 Mem
- Why does this make sense?
  - Look at instruction frequency and common effort
    - » Branch average about every 6 instructions so need that
    - » LD + ST about every 6 as well
    - » seldom need FP Mul & Divide on same cycle
    - » FP Add/Sub share exponent normalization
    - » Int-Divide is done on the FP-Div unit

Dynamic Issue

- Until Now
  - instructions have been issued in order
    - » compiler thinks the world is sequential
    - » HW must fulfill that contract
  - e.g. Issue Queue
- Dynamic Issue
  - basics
    - » use instruction window/buffer rather than a Q
    - » choose the <= n instructions to issue
      - such that dependencies are satisfied
      - and structural rules are not violated
  - 2 methods
    - » control centric: Scoreboarding
    - » data centric: Tomasulo (text focus)

Dynamic Issue Context

- Less viable in multi-core land
  - single thread performance is not longer the Holy Grail
  - power wall is the fundamental constraint
    - » dynamic issue consumes a lot of power
    - » all the OOO/ROB stuff consumes a lot of power
  - thermal wall is also an issue
    - » frequency derating is common
    - » affects reliability & cost in a major way
  - with billion transistor chips
    - » if they're all active then the chip melts
    - » interesting stat in a recent talk
      - C0 state is in play a very small percentage of the time
- Hence
  - I previously spent a lot of time on this issue
    - » this term we'll look at the conceptual side
      - and skip the minutiae

Trends

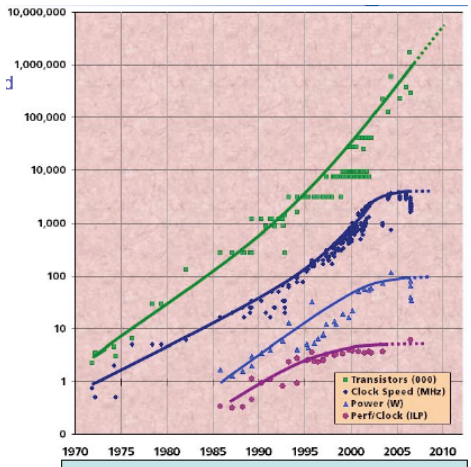
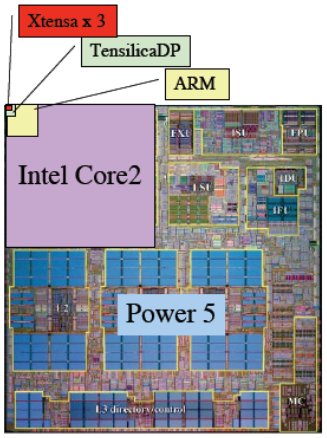


Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith

Core Comparison

- **source: presentation by John Shalf @ NERSC**



- Power5 (Server)
  - 389mm^2
  - 120W@1900MHz
- Intel Core2 sc (laptop)
  - 130mm^2
  - 15W@1000MHz
- ARM Cortex A8 (automobiles)
  - 5mm^2
  - 0.8W@800MHz
- Tensilica DP (cell phones / printers)
  - 0.8mm^2
  - 0.09W@600MHz
- Tensilica Xtensa (Cisco router)
  - 0.32mm^2 for 3!
  - 0.05W@600MHz

Office of Science U.S. DEPARTMENT OF ENERGY

Each core operates at 1/3 to 1/10th efficiency of largest chip, but you can pack 100x more cores onto a chip and consume 1/20 the power

Another Viewpoint

- source: John Shalf

	Traditional Core	Throughput Core	
uArch	Out of Order	In Order	
Size	50	10	mm^2
Power	37.5	6.25	W
Freq	4	4	GHz
Threads	2	4	
Single Thread	1	0.3	Relative Performance
Vector	4 (128-bit)	16 (512-bit)	
Peak Throughput	32	128	GFLOPS
Area Capacity	0.6	13	GFLOPS/mm
Power Capacity	0.9	20	GFLOPS/W

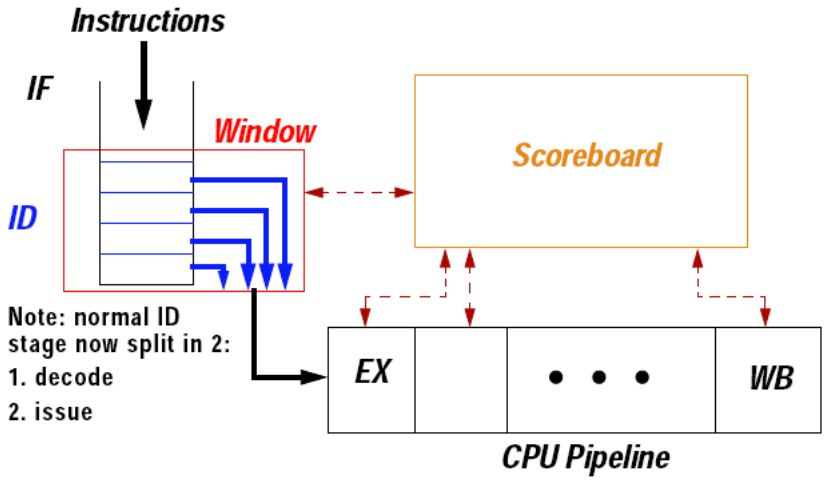
Note: these numbers are a bit optimistic but the trend is correct

Scoreboarding

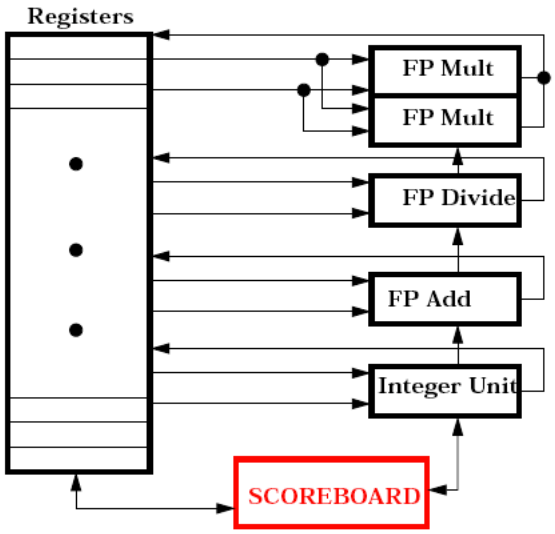
- Introduced by Seymour Cray in the CDC 6600 circa 1964
  - 4 FPUs, 5 MMUs, 7 Ius
  - centralized control knows all
  - RISC like instruction set
    - » 60% performance gain from dynamic reordering
    - » inflated cost by 60% - good thing at $1.2M
    - » not a chip
      - so chip heat and cooling was for the room not the chip
- Later MIPS, IBM, & HP bring it back in single chip guise
  - later changed to more decentralized approach due to long wire phenomenon
  - Alpha was the last to convert to dynamic issue but was short lived
    - » DEC dies and Intel buys the part that is Alpha
      - and then squashes it

Scoreboard Idea

- **Simple in concept, hairy in practice**



Multi-XU Scoreboard



Note: this model could support both single or multi-issue

Exception is that one multiply will be issued per cycle

All depends on bus/trunk structure

Scoreboard contains ROB

Not Shown

- Memory ops
  - scoreboard views memory interface as just another XU
- Branches
  - scoreboard tracks branch resolution
    - » nullifies any speculative instructions in the branch delay slots
- Details of what the scoreboard entries contain
  - similar to the ROB
  - difference is centralized control
    - » gets signals from everywhere and sends enables/selects back
    - » round trip over long wires is prohibitive today for single core
      - note it would work for small cores
      - but it consumes too much power
    - » jury still out whether this is a dead tactic or not

Data Centric Dynamic Issue

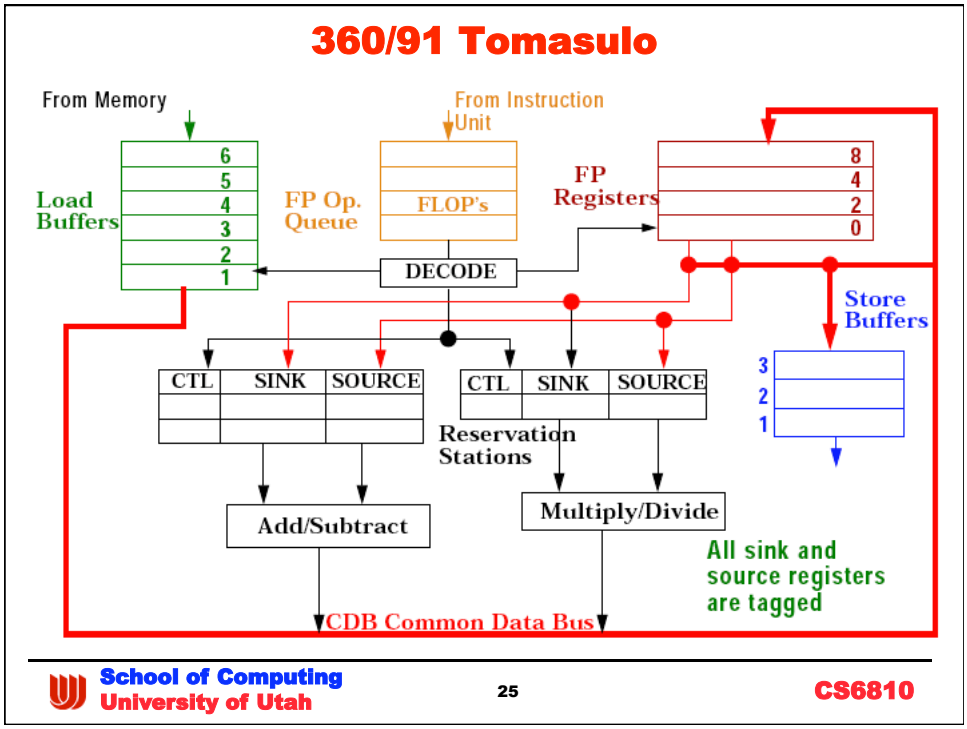
- Started with the IBM 360/91 circa 1969
  - Tomasulo original idea applied only to the floating point units
    - » note:
      - no caches, few registers, no precise exceptions
    - » long and variable cycle latencies
      - note w/o caches operands came from registers or main memory
        - memory was based on ferrite cores
        - dinosaurs were a problem in the parking lot
      - results in out of order completion
    - » note these characteristics now apply to other pieces of the machine
      - memory hierarchy creates unknown latency returns
      - floating point ops still have variable latencies
- Same basic idea but dataflow based
  - dynamic issue and hazard control is still the goal

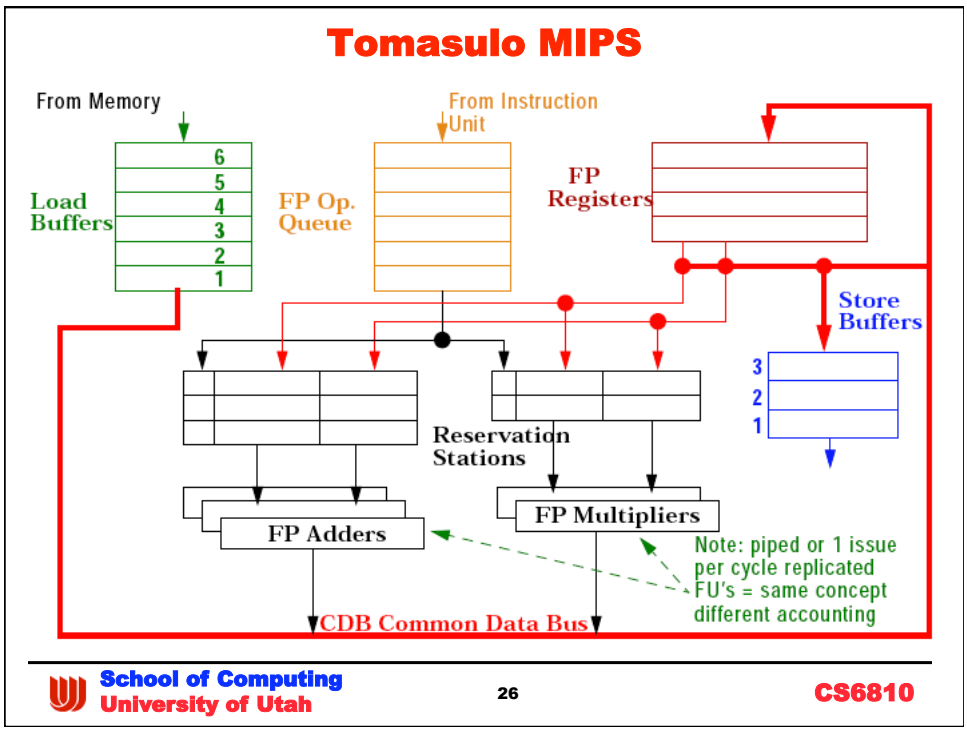
Different Control Model

- Multiple XU's
  - fronted by *"reservation stations"*
  - when reservation station gets all of it's operands the instruction issues into the associated XU
    - » out of order issue & out of order completion
    - » basically a mechanism for implementing data-flow
      - which is the true semantic contract
  - XU's create results which are tagged with the appropriate reservation station slot ID
    - » equivalent of forwarding logic
      - implicitly removes RAW hazards
    - » values placed on a *"common data bus"*
    - » reservation station slots are registers
      - implicit renaming
      - removes WAx hazard problem
- Separate load and store Q's
  - deals with the memory dismbiguation issue
    - » provides a write buffer (we'll see more of this later)

New Pipeline Model

- Fetch
  - in order into instruction queue
- Dispatch
  - in-order into an available reservation station
- Issue
  - happens when a res. station slot gets all of it's operands
    - » instruction packet goes into Execute
- Mem & WB are concurrent
  - makes sense since only LD & ST use the MEM stage anyway
  - WB goes to waiting reservation stations, registers, or memory
- Key point
  - in-order fetch and dispatch
  - out of order completion and issue


360/91 Tomasulo
From Memory
From Instruction Unit
Load Buffers
6
5
4
3
2
1
FP Op. Queue
FLOP's
FP Registers
8
4
2
0
DECODE
Store Buffers
3
2
1
CTL
SINK
SOURCE
CTL
SINK
SOURCE
Reservation Stations
Add/Subtract
Multiply/Divide
All sink and source registers are tagged
CDB Common Data Bus
School of Computing
University of Utah
25
CS6810


Tomasulo MIPS
From Memory
From Instruction Unit
Load Buffers
6
5
4
3
2
1
FP Op. Queue
FP Registers
Store Buffers
3
2
1
Reservation Stations
FP Adders
FP Multipliers
Note: piped or 1 issue per cycle replicated FU's = same concept different accounting
CDB Common Data Bus
School of Computing
University of Utah
26
CS6810

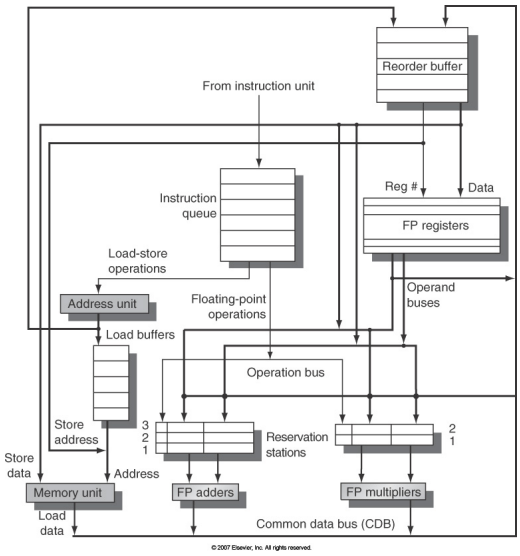
Tomasulo Comments

- CDB is the weak link
  - needs to be wide enough to hold multiple XU results
  - same laminarity issue with a width wrinkle
    - » if you need to execute n instructions/cycle on average
      - fetch, dispatch, issue, CDB needs to support n as well
- Locality
  - layout has surprisingly local wires
    - » no long wire round-trip as per scoreboard approach
  - exception
    - » CDB goes EVERYWHERE
      - power hog and a frequency barrier
        - – high-C multi-drop bus has signal integrity and delay issues
          - – fixed with repeaters but adds delay and power

Tomasulo Memory Issues

- Out of order loads and stores possible
  - OK if addresses don't match
- Dynamic memory disambiguation
  - stall loads if a pending store to the same address
  - OR garner the value from the store unit
  - stall stores when there is a pending load from a previous instruction
- But what about speculation & exceptions
  - note exceptions weren't precise in 1969
    - » as far as I can tell nothing was
    - » famous Wavy Gravy comment
      - "If you can remember the 60's you weren't there"
  - Add the ROB?
    - » it worked before and it will again

Tomasulo + ROB



Final Comments

- This stuff is important conceptually
  - advise to go through the exercises in the text
    - » the mid-term will definitely have something similar
  - BUT
    - » don't get too whacked on their particular algorithm
      - others exist
  - some aspect of these ideas are likely useful in the future
    - » for now instructions in a thread are sequential
      - HW dynamics can help exploit ILP
  - key issue
    - » socket/chip these days has multiple cores
    - » maximizing performance/watt is a critical concern
      - speculation and HW dynamics can be too *"watty"*
        - but there's no need to go crazy at least right now
        - and in the foreseeable future