VLIW Processors

- Package multiple operations into one instruction
- Example VLIW processor:
  - One integer instruction (or branch)
  - Two independent floating-point operations
  - Two independent memory references
- Must be enough parallelism in the code to fill the available slots

VLIW Processors

- Disadvantages:
  - Statically finding parallelism
  - Code size
  - No hazard detection hardware
  - Binary code compatibility

VLIW Example

Source instruction	Instruction using result	Latency
FP ALU OP	FP ALU OP	3
FP ALU OP	Store double	2
Load double	FP ALU OP	1
Load Double	Store double	0

```
Loop: L.D      F0,0(R1)
      ADD.D    F4,F0,F2
      S.D      0(R1),F4
      DADDUI   R1,R1,#-8
      BNE  R   1,R2,Loop
```

For (I=1000;I>0;I++)

x[I]=x[I]+s;

VLIW Example

- Assume that w can schedule 2 memory operations, 2 FP operations, and one integer or branch

Memory reference 1	*Memory reference 2*	*FP operation 1*	*FP op. 2*	*Int. op/ branch*	*Clock*
LD F0,0(R1)	LD F6,-8(R1)				1
LD F10,-16(R1)	LD F14,-24(R1)				2
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2		3
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2		4
		ADDD F20,F18,F2	ADDD F24,F22,F2		5
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2			6
SD -16(R1),F12	SD -24(R1),F16			DADD R1,R1,#-56	7
SD 24(R1),F20	SD 16(R1),F24				8
SD 8(R1),F28				BNEZ R1,LOOP	9

Dynamic Scheduling, Multiple Issue, and Speculation

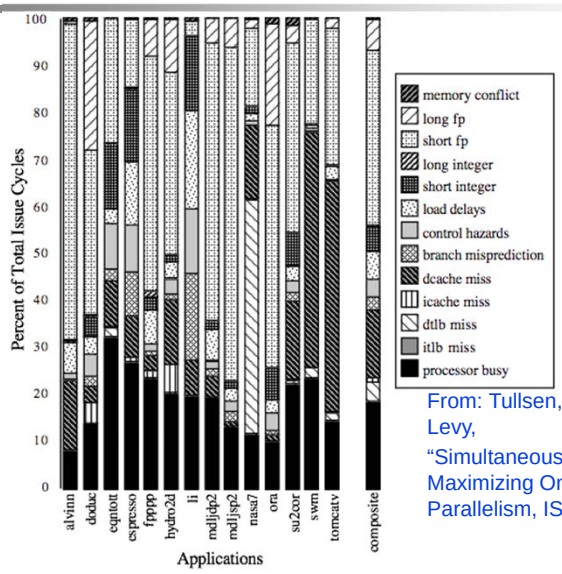
- Modern microarchitectures:
  - Dynamic scheduling + multiple issue + speculation
- Two approaches:
  - Assign reservation stations and update pipeline control table in half clock cycles
    - Only supports 2 instructions/clock
  - Design logic to handle any possible dependencies between the instructions
  - Hybrid approaches
- Issue logic can become bottleneck

Multiple Issue

- Limit the number of instructions of a given class that can be issued in a “bundle”
  - I.e. one FP, one integer, one load, one store
- Examine all the dependencies among the instructions in the bundle
- If dependencies exist in bundle, encode them in reservation stations
- Also need multiple completion/commit

Example

```
Loop:  LD R2,0(R1)        ;R2=array element
       DADDIU R2,R2,#1    ;increment R2
       SD R2,0(R1)        ;store result
       DADDIU R1,R1,#8    ;increment pointer
       BNE R2,R3,LOOP     ;branch if not last element
```

From: Tullsen, Eggers, and Levy,
"Simultaneous Multithreading: Maximizing On-chip Parallelism, ISCA 1995.

Thread Level parallelism

- Multithreading: multiple threads to share the functional units of 1 processor via overlapping
  - processor must duplicate independent state of each thread e.g., a separate copy of register file, a separate PC, and for running independent programs, a separate page table
  - memory shared through the virtual memory mechanisms, which already support multiple processes
  - HW for fast thread switch; much faster than full process switch ≈ 100s to 1000s of clocks
- When to switch?
  - Alternate instruction per thread (fine grain)
  - When a thread is stalled, perhaps for a cache miss, another thread can be executed (coarse grain)

Fine-Grained Multithreading

- Switches between threads on each instruction, causing the execution of multiples threads to be interleaved
- Usually done in a round-robin fashion, skipping any stalled threads
- CPU must be able to switch threads every clock
- Advantage is it can hide both short and long stalls, since instructions from other threads executed when one thread stalls
- Disadvantage is it slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from other threads
- Used on Sun's T1

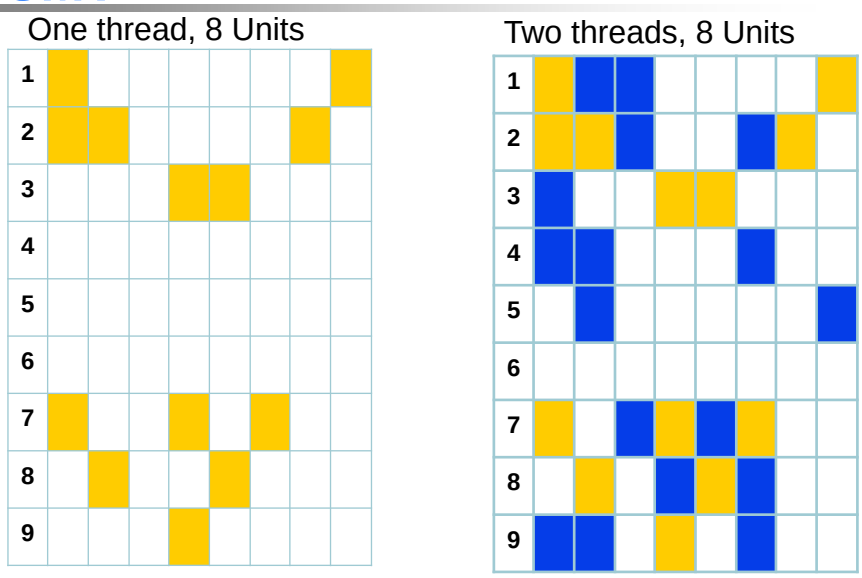
Coarse-Grained Multithreading

- Switches threads only on costly stalls, such as L2 cache misses
- Advantages
  - Need to have very fast thread-switching
  - Doesn't slow down thread, since instructions from other threads issued only when the thread encounters a costly stall
- Disadvantage is hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs
  - Since CPU issues instructions from 1 thread, when a stall occurs, the pipeline must be emptied or frozen
  - New thread must fill pipeline before instructions can complete
- Because of this start-up overhead, coarse-grained multithreading is better for reducing penalty of high cost stalls, where pipeline refill << stall time

Simultanuous Multithreading SMT

- Fine-grained multithreading implemented on top of multiple-issued dynamically scheduled processor.
- Multiple instructions from different threads.

SMT



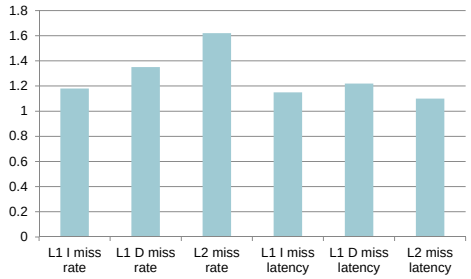
Multithreading

Sun T1

- Focused on TLP rather than ILP
- Fine-grained multithreading
- 8 cores, 4 threads per core, one shared FP unit.
- 6-stage pipeline (similar to MIPS with one stage for thread switching)
- L1 caches: 16KB I, 8KB D, 64-byte block size (misses to L2 23 cycles with no contention)
- L2 caches: 4 separate L2 caches each 750KB. Misses to main memory 110 cycles assuming no contention

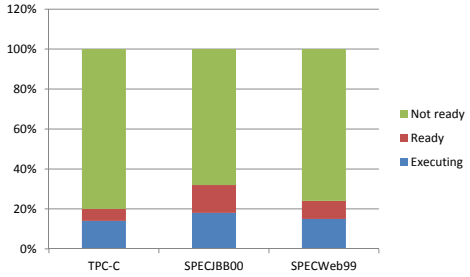
Sun T1

- Relative change in the miss rate and latency when executing one thread per core vs 4 threads per core (TPC-C)



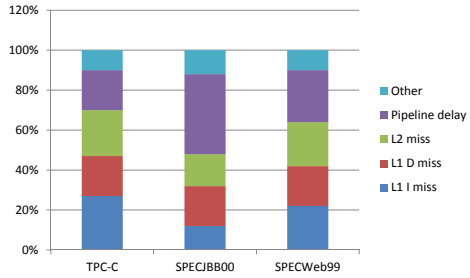
Sun T1

- Breakdown of the status on an average thread. Ready means the thread is ready, but another one is chosen – The core stalls only if all the 4 threads are not ready



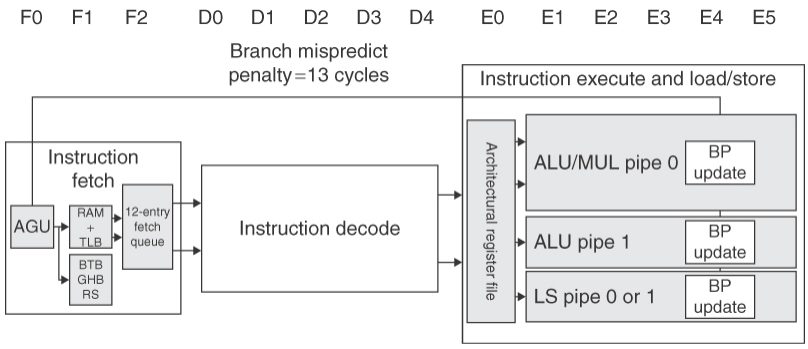
Sun t1

- Breakdown of the causes for a thread being not ready



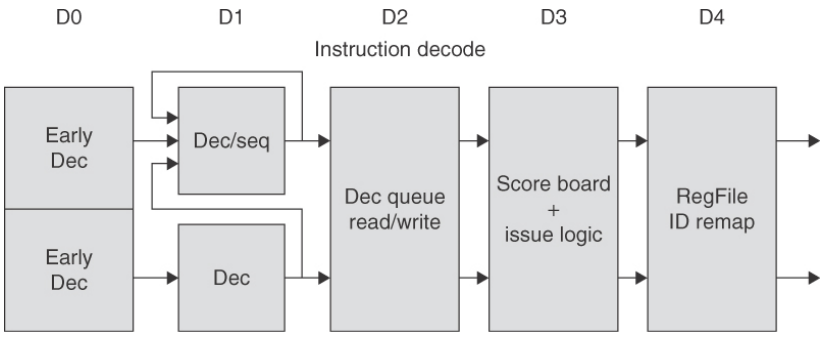
The ARM Cortex-A8

- Dual issue processor 13-stage pipeline



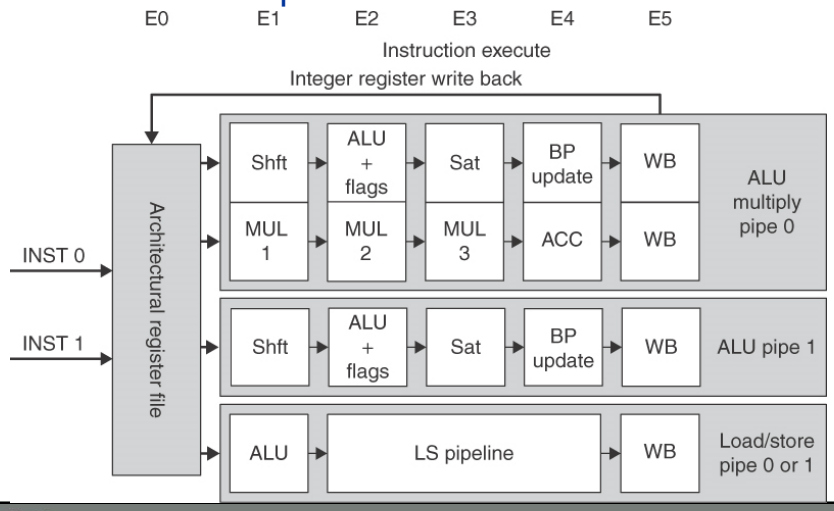
The ARM Cortex-A8

- Five stage instruction decode



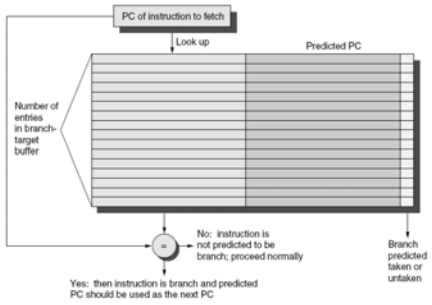
ARM Cortex-A8

- Execution Pipeline



Branch-Target Buffer

- Need high instruction bandwidth!
  - Branch-Target buffers
    - Next PC prediction buffer, indexed by current PC



Branch Folding

- Optimization:
  - Larger branch-target buffer
  - Add target instruction into buffer to deal with longer decoding time required by larger buffer
  - “Branch folding”

Return Address Predictor

- Most unconditional branches come from function returns
- The same procedure can be called from multiple sites
  - Causes the buffer to potentially forget about the return address from previous calls
- Create return address buffer organized as a stack

Integrated Instruction Fetch Unit

- Design monolithic unit that performs:
  - Branch prediction
  - Instruction prefetch
    - Fetch ahead
  - Instruction memory access and buffering
    - Deal with crossing cache lines

Register Renaming

- Register renaming vs. reorder buffers
  - Instead of virtual registers from reservation stations and reorder buffer, create a single register pool
    - Contains visible registers and virtual registers
  - Use hardware-based map to rename registers during issue
  - WAW and WAR hazards are avoided
  - Speculation recovery occurs by copying during commit
  - Still need a ROB-like queue to update table in order
  - Simplifies commit:
    - Record that mapping between architectural register and physical register is no longer speculative
    - Free up physical register used to hold older value
    - In other words: SWAP physical registers on commit
  - Physical register de-allocation is more difficult

Integrated Issue and Renaming

- Combining instruction issue with register renaming:
  - Issue logic pre-reserves enough physical registers for the bundle (fixed number?)
  - Issue logic finds dependencies within bundle, maps registers as necessary
  - Issue logic finds dependencies between current bundle and already in-flight bundles, maps registers as necessary

How Much?

- How much to speculate
  - Mis-speculation degrades performance and power relative to no speculation
    - May cause additional misses (cache, TLB)
  - Prevent speculative code from causing higher costing misses (e.g. L2)
- Speculating through multiple branches
  - Complicates speculation recovery
  - No processor can resolve multiple branches per cycle

Energy Efficiency

- Speculation and energy efficiency
  - Note: speculation is only energy efficient when it significantly improves performance
- Value prediction
  - Uses:
    - Loads that load from a constant pool
    - Instruction that produces a value from a small set of values
  - Not been incorporated into modern processors
  - Similar idea--*address aliasing prediction*--is used on some processors