COS375
Fall 2023

github/jovyntls github/JothamWong

Instruction Set Architecture

- Architecture: Abstraction layer provided to SW to interface with HW.
  - Examples: ARM, x86, MIPS
  - Designed for stability and to hide implementation details.
  - Includes programmer visible state/operations/instructions/data types/sizes/execution semantics.
- Microarchitecture: Physical instance of contract. Implementation details.
  - Includes additional state not visible to programmer such as cache size.
- Design goals
  1. Portability
  2. Programmability
  3. Efficiency (power/energy/speed/code size/cost)

MIPS

- MIPS: Microprocessor without Interlocked Pipelined Stages
- word size: 32 bits or 4 bytes (1 byte = 8 bits)
- Branching in MIPS has delay slots: the next instruction after a branch command happens before the PC is updated to the branch

Data/Control Flow Graph

- Basic block: Single entry/single exit. You cannot execute instructions in the middle without executing from the start.
- Liveness: A register is live at a point in code if it holds a value that may be needed later.
  - All reads expose a live range for that register upward.
  - All writes prevent a live range for that register from continuing upward.
- Live out registers: registers who values are used afterwards. Cannot be overridden.
- Data dependency: Three forms of dependencies
  - Flow/RAW (True dependency):
  - Output/WAW (False dependency):
  - Anti/WAR (False dependency):
- False dependency == can be removed with register renaming
- Dominance: A node (basic block or instruction) M dominates a node N if every path in the CFG from entry to N passes through node M.
- All nodes dominate themselves, so the entry node dominates all nodes, including itself.
- Optimization tricks
  - Register renaming
  - Constant propagation: The constant of a move immediate operation replaces later uses of the destination register.
    - Safe to perform when 1) move-immediate dominates the use, 2) no intervening writes of the destination register on any path between the move-immediate to the use.
  - Constant Folding: Constants within an ALU instruction can be replaced with the computed value. Always safe to perform.
  - E.g: r1 = 3 + 5 becomes r1 = 8
  - Dead Code Elimination: Instructions that only write to dead registers (dead == value not used) can be eliminated. If an instruction writes to a dead register but has side effects (throws exceptions, modifies memory), it is not dead code.
- Sign Extension: Copy the most significant bit until 32 (or whatever) bits. Returns a signed integer.
- Zero Extension: Pad up with 0s until 32 (or whatever) bits. Returns an unsigned integer.

Integer representation

Bit Pattern	Unsigned	Sign Magnitude	One's Complement	Two's Complement
000	0	+0	+0	0
001	1	+1	+1	+1
010	2	+2	+2	+2
011	3	+3	+3	+3
100	4	-0	-3	-4
101	5	-1	-2	-3
110	6	-2	-1	-2
111	7	-3	-0	-1

- Sign magnitude and one's complement: Have duplicate zeros
- Two's complement: most significant bit is sign bit, 1 if negative 0 if positive
- The sign bit is multiplied by -1 and the rest are positive to get value
- Negating a two's complement: invert all bits then add 1 to result
- Sign extension works because 2 complement's really have infinite 0/1s on the left if positive/negative
- Overflow: With w bits, overflow is just (u + v) mod2^w
  1. Hardware does not detect overflow (MIPS: addu, addiu, subu). Onus is on software to check.
  2. Hardware maintains a flag to detect overflow but does nothing.
  3. Hardware raises exception, PC jumps to predefined addr for exception and prior addr is saved for possible recovery. (MIPS: add, addi, sub)

Pseudo Instructions

Name	Example	Equivalent MISP Instructions

Load address	`la $t0, label`	`lui $at, hi-16-bits`
		`ori $t0, $at, lower-16-bits`

Load immediate	`li $t0, 0xdeadbeef`	`lui $at, 0xdead`
		`ori $t0, $at, 0xbeef`

Branch if less or equal	`ble $s0, $s1, label`	`slt $at, $s1, $s0`
		`beq $at, $zero, label`

Move	`move $s0, $s1`	`add $s0, $s1, $zero`
No operation	`nop`	`sll $zero, $zero, 0`

at (register no 1) is the assembler temporary and is reserved by the assembler.

Control instructions

- In assembly, the branch is typically short distance.
- Jump addr can be anywhere.

Addressing Modes

- Memory can be viewed as a single 1-dimensional array of words (32 bits/4 bytes).
- MIPS words are aligned: this means we need to times 4 when indexing (see later).

Addressing Modes AND CISC/RISC Review

Immediate	r1 = r2 + 5	
Register	r1 = r2 + r3	
PC Relative	branch r1 < r3, 1000 (-40)	
Direct	r1 = M [4000]	33% avg, 17% to 43%
Register Indirect	r1 = r2 + M[r2]	13% avg, 3% to 24%
Displacement	r1 = M[r2 + 4000]	42% avg, 32% to 66%
Memory Indirect	r1 = M[M[r2]]	3% avg, 1% to 6%
Scaled	r1 = M[100 + r3 + r4 * d]	7% avg, 0% to 16%
Indexed/Base	r1 = r3 + M[r2 + r3]	2% avg, 0% to 3%
Autoincrement	r1 = M[r2 + 4000] (r2+=d)	2% avg, 0% to 3%
Other		2% avg, 0% to 3%

- Indexed/base, scaled, and memory indirect not supported in MIPS

Endianness

How are bytes of a word laid out?

➢ **Little Endian:** Least-significant byte is stored in the location with the lowest address (little end first)

Address	0000	0001	0002	0003
Byte #	0	1	2	3

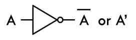
➢ **Big Endian:** Most-significant byte is stored in the lowest address (big end first)

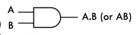
Address	0000	0001	0002	0003
Byte #	3	2	1	0

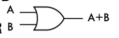
Registers vs Memory

- Registers are faster to access than memory as operating on memory requires load and store ops but have comparatively little space.
- Space available
  - Registers: 32 32 bits registers = 32 * 4 bytes = 2^7 bytes
  - Memory: 2^{32} bytes (2^{25} times more space)
- Compilers use registers as much as possible, spill to memory when required (known as register pressure).

Boolean Algebra

- NOT 

- AND 

- OR 

- Boolean Algebra Laws
  1. Identity : A.1 = A, A + 0 = A
  2. Zero and one: A.0 = 0, A + 1 = 1
  3. Inversion : A.$\overline{A}$ = 0, A + $\overline{A}$ = 1
  4. Idempotence : A.A = A, A + A = A (Useful for generating extra terms)
  5. Commutativity: A.B = B.A, A + B = B + A
  6. Associativity: A.(B.C) = (A.B).C, A + (B + C) = (A + B) + C
  7. Distribution: A + (B.C) = A.B + A.C, A + (B.C) = (A + B).(A + C)
  8. DeMorgan's : $\overline{A.B}$ = $\overline{A}$ + $\overline{B}$, $\overline{A+B}$ = $\overline{A}$. $\overline{B}$

Single Cycle Processor

- Cycle

RISC vs CISC

Performance

- Million Instructions Per second (MIPS) = $\frac{\text{Instruction count}}{\text{Execution Time} \times 10^6}$
- The Iron Law of Computer Performance:

$$\text{Time} = \text{Instructions} \times \frac{\text{Cycles}}{\text{Instructions}} \times \frac{\text{Time}}{\text{Cycles}}$$

- $\frac{\text{Cycles}}{\text{Instructions}}$ = CPI (1 / IPC)
- $\frac{\text{Time}}{\text{Cycles}}$ = 1/(Clock Frequency)

Category	Example	Instruction Count	CPI	Cycle Time (1/frequency)
Algorithm	Sort	✓	✓	✗
Programming Language	C, Python	✓	✓	✗
Compiler	GCC → LLVM	✓	✓	✗
ISA	MIPS → x86	✓	✓	✓
µArch		✗	✓	✓
Technology		✗	✗*	✓

- * If memory takes the same amount of time, loads might have to wait more cycles – CPI would change in that case

Index Card Quiz

$$Time = Instructions \times \frac{Cycles}{Instruction} \times \frac{Time}{Cycle}$$

You are designing a new CPU and a new ISA. The base design has 32 general-purpose registers and operates at a clock rate of 1GHz. A member of your CPU's compiler team has suggested that if you can architect in 64 registers instead of 32 (i.e., change the instruction encodings, etc.), they will be able to reduce instruction counts for a key target program from 10M to 9M. But you suspect that the larger register file will be slower to access, and so the cycle time will get worse. What is the "break-even" cycle time and clock rate for this?

NOTE: 1GHz (1 billion cycles per second) = 1/1ns (each cycle is 1 nanosecond or 1 billionth of a second)
<u>Also Associate:</u>
MHz and microsecond
KHz and millisecond
Hz and second

$9M \cdot \cancel{CPI} \cdot n = 10M \cdot \cancel{CPI} \cdot 1$

$1.111 \frac{ns}{cycle}$ $\qquad \frac{1}{1.111 \frac{ns}{cycle}} = .9$ GHz or 900 MHz

-

Multicycle Processor

Pipelining