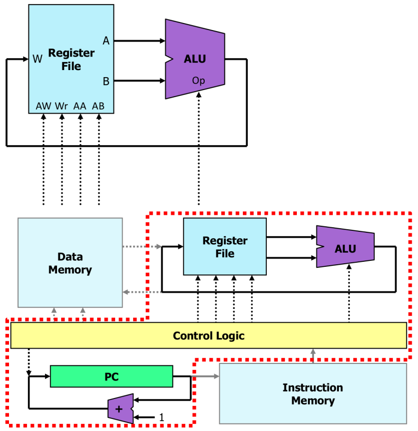
1 Processor

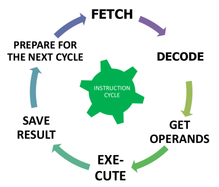


2 ISA

Instruction classes + registers + addressing mode + memory organisation/access + instruction encoding

CISC : few registers, complex and variable size instruction encoding, complex addressing modes, variable execution pattern (Intel)
RISC : 2^n equivalent registers, regular (1 or 2 sizes) instruction encoding, load/store architectures, regular execution pattern (MIPS, Alpha, most recent ISAs)

Compiler : code $\rightarrow$ assembly
Assembler : assembly $\rightarrow$ machine code



2.1 Flow

Jumping : unconditionally changing flow
Branching : conditionally changing flow
Calling functions / returning from functions

2.2 MIPS

32 registers (32-bit wide)
Memory in bytes, load/store words aligned to a word boundary

Endianness : order in which bytes are arranged into words (MIPS is bi-endian)

2.2.1 Functions

Caller : making the call
Callee : taking the call

Caller-saved registers : responsibility of the caller, caller pushes them on stack before function call, caller pops their original value from stack after returning from callee, callee can use them freely
Callee-saved registers : responsibility of callee, callee pushes them on stack before modifying them, callee pops their original value from stack before returning to the caller, caller safely assumes that registers are not permanently modified by callee

Not enough registers : stack (last-in, fist-out), grows towards low addresses

3 Computer Arithmetic

3.1 Integer

Unsigned ($\geq$ 0) binary number :
$A = \langle a_{n-1}, \dots, a_0 \rangle = \sum_{i=0}^{n-1} a_i 2^i$
can be seen as having infinite leading zeros
Sign and Magnitude (S&M) :
$A = \langle s\, a_{n-1}, \dots, a_0 \rangle = (-1)^s \cdot \sum_{i=0}^{n-2} a_i 2^i$
Leftmost bit represents sign (1 for negative)
Familiar for users, simple multiplication, not efficient adders, redundant zero (± 0)
Two's complement : $A = \langle a_{n-1}, \dots, a_0 \rangle = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$
$A_{\text{max}} = 2^{n-1} - 1, A_{\text{min}} = -2^{n-1}$
sign extension : add leading zeros (ones) for positive (negative) numbers
$A + \overline{A} = -1$ so $-A = \overline{A} + 1$

Overflow : makes sum of positives negative or sum of negatives positive
`addu` does not check overflow, `add` stops execution in case of overflow
Multiplication on MIPS : result into special register pair `Hi` & `Lo`, overflow not tested move results back into regular registers `mflo` `mfhi` move from low/high

3.2 Fixed-Floating point

3.2.1 Fixed point

Adds a fractional point in a fixed position, m binary digits past point, n binary digits left of the point :
$A_{n,m} = \langle a_{m+n-1} \dots a_m . a_{m-1} \dots a_0 \rangle$
$= \frac{1}{2^m} \sum_{i=0}^{m+n-1} a_i 2^i$
Very cheap and fast, portable (every processor has integer processing units), costs lie in accuracy

3.2.2 Floating point

Two parts (each positive or negative) : M mantissa (significand) normalized, E exponent field
$X = M \cdot 2^E$
Large dynamic range, bad resolution (minimum difference between two successive floating point numbers), bad precision
Floating point is not $\mathbb{R}$: $(a+b)+c \neq a+(b+c)$

Biased representation :
$A = \langle a_{n-1} ... a_0 \rangle = \sum_{i=0}^{n-1} a_i 2^i - B$

IEEE 754 : hidden bit is not stored (actual value of mantissa is $1.M$, be careful for 0), exponent is biased representation $B = 127$ for single precision, $B = 1023$ for double precision
$X = 1.M \cdot 2^{E-B}$
Sigle precision range : $\pm 2 \cdot 10^{-38} \rightarrow 2 \cdot 10^{38}$
Double precision range : $\pm 2 \cdot 10^{-308} \rightarrow 2 \cdot 10^{308}$
Zero values are represented by the biased exponent and mantissa both being 0, sign bit does not matter (± 0)
Positive and negative infinity (result of arithmetic overflow) : biased exponent equal to all 1, faction equal to all 0, sign for $\pm\infty$
NaN (not a number) result of invalid operations : biased exponent equal to all 1, fraction equal to anything but all 0, sign bit does not matter
rounded to nearest value with a 0 LSB
FLAGS : overflow ∞, underflow, division by 0 ∞, inexact result, invalid operation Nan

ISA : floating point is rare in embedded processor, emulated in software, often dedicated separate register file and special instructions for floating point
MIPS : floating-point coprocessor, own registers \$f0-\$f31 32-bit wide, different instructions depending on the precision

4 Cache memory

SRAM (cache) : content last as long as there's power, (like flip flop), low density, high power, expensive, fast
DRAM (memory) : needs to be refreshed regularly, (a capacitor and a switch, one transistor), high density, low power, cheap, slow
Temporal locality : data used recently, high likelihood of being use again (loop, functions, local variables)
Spatial locality : data currently used are likely to be accessed in the future (arrays, sequential elements)
Cache design : holds most recently accessed data/instructions, divided into units (cache blocks with cache tag holding the main memory address)
Hit and miss : hit = data is found in cache, miss = data not found, hit (miss) rate = number of hits (misses) over total number of cache accesses
Cache tag : every block has address tag keeping (part of) main memory address of tha data stored

Read hit : main memory not accessed, cache replies sending requested memory content, quick
Read miss : accesses main memory to get data, data stored (allocated) in cache and passed to processor, longer waiting time
Write hit write-through : data written both into cache and main memory, straightforward, keep the memory/buses busy for nothing, good for write and re-read data frequently, low read latency
Write hit write-back : data updated only in cache (main memory will eventually become wrong/obsolete), good for write-intensive applications, remember which blacks are obsolete (additional bit, dirty bit), before overwriting dirty block from cache its content is stored to memory, slow if many read misses, best for mixed (read and write) workloads
Write miss allocate : allocate the data block in cache, signal write hit, write-back caches use write-allocate
Write miss no-allocate : write data to memory, not allocate in cache, wait next read miss to load data and allocate in cache, write-through caches use write no-allocate
Eviction policies : no space for new data, must overwrite one of the cache lines (evection/replacement), least recently used vs random
Associativity : number of cache lines where one word of data can be placed

4.1 Fully associative cache

A word can go in whichever line of the cache (associativity of L for cache with L lines), compare incoming tag with all existing tags in parallel, huge logic overhead, very small capacity, high hit rate
Valid bit : initial cache content is garbage, valid bit added to every cache line to indicate whether content is relevant, at start/reset valid bit is 0, hit only if tag matches and valid bit is 1
Temporal locality : one word per line, load from main memory to cache exclusively required data and when required, not bringing data in advance
Spatial locality : multiple words per line, load not only required word but also words near it
If 2^m words are loaded at a time and memory address is n bits wide, last m bits of the address used to select a word from cache line and first $(n-m)$ bits of address form the tag of cache line

4.2 Direct Mapped cache

A word is mapped to a single line of the cache, associativity of 1
Cache capacity of 2^m words, m address bits used to index cache lines, $(n-m)$ address bits for a tag, each address directly maps to a single cache line, incoming tag compared with only one cache tag, behaving like standard memory, low hit rate
Alias : different addresses use same line of the direct mapped cache, resulting in frequent cache pollution or conflict misses

4.3 Set-associative cache

Every word is mapped to as many cache lines/blocks as there are ways, (called n-way set-associative), intermediate hit rate
Number of sets = number of lines over number of ways, number of address bits to select words inside a line = $\log_2(N_{\text{ways}})$, number, number of address bits to index sets $\log_2(N_{\text{sets}})$

4.4 Performance

Measure of performance : execution time of real programs
X n times faster than Y : $n = \frac{\text{Y execution time}}{\text{X execution time}}$
Amdahl's law : Speedup $= \frac{\text{Execution time without enhancement}}{\text{Execution time using enhancement}}$
Execution time$_{\text{new}}$ $=$ Execution time$_{\text{old}}$ $\times ((1 -$ Fraction$_{\text{enhanced}}) + \frac{\text{Franction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}})$
Speedup$_{\text{overall}}$ $= \frac{\text{Exectution time}_{\text{old}}}{\text{Execution time}_{\text{new}}}$

Instruction count IC : instructions per program, determined by ISA and compiler
Clock Cycles per instruction CPI, determined by organization and ISA
Clock Cycle time : seconds per clock cycle, $\frac{1}{\text{clock freq}}$, determined by hardware and organization
CPU time : time a CPU takes to execute a program $(IC \times CPI \times CCT)$

4.5 Hierarchy

Memory hierarchy organized in several levels, each more expensive per byte than next level farther from processor
Word not found in cache, must be fetch from next level, multiple word move at a time
Miss rate : misses per memory references, misleading because do not factor in cost of a miss
Average memory access time : Hit time $+$ Miss rate $\times$ Miss penalty
Two-level cache : Hit time$_1$ $+$ Miss rate$_1$ $\times$ (Hit time$_2$ $+$ Miss rate$_2$ $\times$ Miss penalty$_2$)

Cache misses : Capacity, Conflict/collision (for direct mapped or set-associative), Compulsory/cold-start/first-reference (first access to block cannot be hit)

4.5.1 Virtual Memory

Program about to run : copied from disk (permanent) storage into main memory DRAM, CPU program counter set to starting address of program
Physical memory (address) : memory actually available in computer (location in physical memory)
Virtual Memory (address) : memory the OS allows a program to believe available (conventional address used by program, os must translate to physical address), cheap (need only as much RAM as program

references), automated memory/disk management, enables multiprogramming time-sharing and protection

MMU : performs dynamic address translation, placed between CPU and cache (cache called physical) or between cache and main memory (cache called virtual)

Program relocation : program written without knowing where stored in main memory

Relocation at load time : done at binary level (not assembly code), challenge is how to find where in binary are addresses, limitations (large binary -> lot of work at load time, inflexible - cannot change later, poor utilization of memory - fragmentation)

Relocation in Hardware on-the-fly : physical address obtained by adding constant (base) to processor generated virtual address, OS changes content of base and bounds registers before passing control to program (context switch), if virtual address is out of bounds irregular situation signalled

Virtual memory Techniques :

Segmentation : base and bounds, splits physical memory exactly as needed by program, arbitrary start address and length of region & contiguous memory addresses

Paging : split physical memory in small blocks of identical size (4-64Kb) called pages (or a frame), assigns as many pages as needed by program, size always power of 2, address within virtual page maps to single location within physical page, starts at address aligned with it size (some of least significant virtual address bits do not need translation), simply hardware (translations kept in main memory, page table)simpler pace assignment (find unused physical page), intrinsic waste (partially unused page), page tables can be large (not linear arrays), at least two memory accesses to get one word (1 read page table, 2 access data/instruction in memory)

Space management : if several programs run at same time, memory shortage might occur and must be handled

Add valid bit to each page table entry indicating if page is in main memory (DRAM) or secondary storage (disk)

Page fault : page is not in main memory, not an error, processor cannot fetch page from disk (MMU asks OS to do it), if main memory full OS evict a page from memory and write to disk (victim page), OS copies requested page from disk to main memory (swapping two pages, slow), in many OSs done in special (swap) partition of disk, swapping takes time, pages much larger than cache blocks, use dirty bit in page table to remember if write was made to the page

Page repacement algorithms : which page to evict, optimal = page that will not bef used (referenced) again or not used for longest period (guarantees lowest possible page-fault rate, cannot be done unless future known), FIFO, LRU, LFU (smallest usage count)

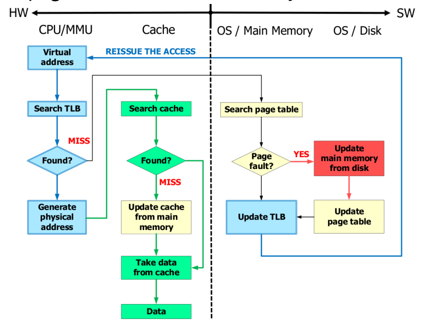
Control bits in page table : valid (page is in main memory or on disk), used (used by evection policy, which page to evict), dirty (indicate a copy-back to disk needed prior to eviction), protection bits (rights to read/write to page), specific for operating system

Memory Protection : programs written to access only their own data, system must protect the programs from unallowed accesses, implicitly enforced by translation controlled by OS

TLB : like small cache, avoid reading page table from main memory on every reference, part of MMU, address-translation cache (stores recent translations), TLB hit (miss) = requested address translation (not) in TLB, placed between CPU and cache, between cache and main memory

Exception handlers : special procedures taking appropriate actions on exception, arithmetic problems (over/underflow) - memory protection violation - input/output device request - use of unsupported undefined instructions - hardware malfunctions (power failures, ...) - tracing instructions (breakpoints, ...), a way for the hardware to invoke software

TLB miss : TLB miss exception, handled in software, OS invoked and walks page table in main memory, missing translation found = bought to TLD and hit on retry, missing translation = page fault and OS brings missing page from dit to main memory and hit on retry



MMU, cache, main memory, disk

4.6 Optimization

Reducing miss rate :

Larger block size : reduce compulsory misses, increased miss penalty, increase conflict and capacity misses

Bigger cache size : reduce capacity misses, potentially longer hit time, higher cost and power

Higher associativity : reduces conflict misses (direct mapped of size N has about same miss rate as two-way set-associative of size $\frac{N}{2}$), may reduce clock time (increase hit time), memory access time grows, increased power

Reduce miss penalty :

Multilevel caches : more power efficient thn single cache

Serve reads before writes completed

Reduce time to hit : small first-level cache, avoid address translation when indexing cache

5 Acronyms

ALU	Arithmetic Logic Unit
PC	Program Counter
ISA	Instruction Set Architecture
CISC	Complex Instruction Set Computer
RISC	Reduced Instruction Set Computer
MIPS	Microprocessor without Interlocked Pipeline Stages
S&M	Sign And Magnitude
NaN	Not a Number
SRAM	Static Random Access Memory
DRAM	Dynamic Random Access Memory
LRU	Least Recently Used
LFU	Least Frequently Used
FIFO	First-in First-out
IC	Instruction Count
CPI	clock Cycles Per Instruction
TLB	Translation Lookaside Buffer
OS	Operating System
MMU	Memory Management Unit