IS1200 Computer Hardware Engineering

Lecture 1

Moore's law - Transistordencity doubles every 18-24 Months
In 2021 thermal limits are the problem. Instead focus is directed to Parallell and multicore processing.

C

C is an imperative language
C is not type safe.
C allows low level memory access

233 - Decimal ['c', 'h', 'a', 'r']
0x1A - Hex "string"
012 - octal 1,501 - float
0b101 - binary

and 01101 & 10111 = 00101
or 01101 | 01001 = 01101

3 << = 12
8 >> = 2

Lecture 2

PRT I

Instruction Set Architecture (ISA)

The ISA is the interface between hardware and software.
For example x86, Arm, Mips.
Two categories:

Complex Instruction Set computer (CISC)

Many special purpose instructions.
↳ x86 has nearly 900 instructions
Various encoding length
↳ x86, 1-15 bytes
Different number of clock cycles for different instructions.

Reduced Instruction Set Computer (RISC)

Few regular functions to minimize complexity
↳ Mips, ARM
Fixed instruction length.
↳ Mips 4 bytes.
One clock cycle per instructions

C to machinecode

C code `a=b+c;` → MIPS assembly code `add $s0 $s1 $s2` → MIPS Machine code `0x02328020`

The c compiler maps variables to registers

Human readable of what the machine sees.

Maps into instructions for the machine to read.

Registers

Name	Number	Use
$0	0	value 0
$at	1	assembler temp
$v0-1	2-3	function return value
$a0-3	4-7	function argument
$t0-7	8-15	temp (caller saved)
$s0-7	16-23	saved (callee saved)
$t8-9	24-25	temp (caller saved)
$k0-1	26-27	OS Kernel
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

MIPS

Has a reference sheet!

and/or/xor/nor $s0, $s1, $s2

andi/ori/xori $s0, $s1, 0xAB41

beq - branch if equal

```
    beq $s1, $t0 foo   # if $s1 == $t0: foo()
    ...
foo:
    add $s5, $s1, $0
```

J - Jump

```
loop:                  # while True:
    ...
    J loop
```

```
loop:                  # while t1 != 0:
    beq $t1 $0 done        s1++
    add $s1
    J loop
done:
    nop
```

Lecture 3

Two's Compliment

In short negative numbers have a leading 1 and are formed by inverting the positive number and adding one.

-7 -> 0111 -> 1000 -> 1001

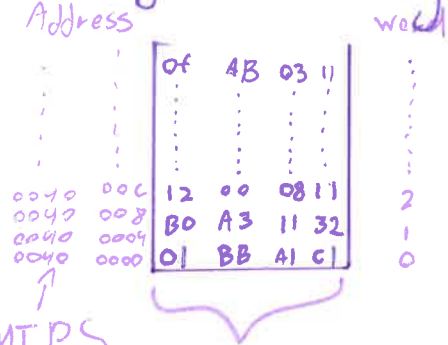
minus 7 in two's compliment

Sign extention:
you must copy the first bit

1010 -> 11111010

0010 -> 00000010

Storage in memory

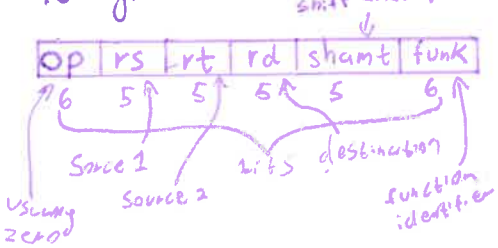


MIPS Programs start at address 0040 0000

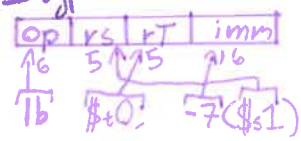
4 bytes 32 bits

All MIPS code is 32 bits long for every instruction.

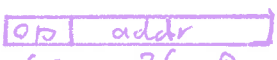
R-Type instruction:



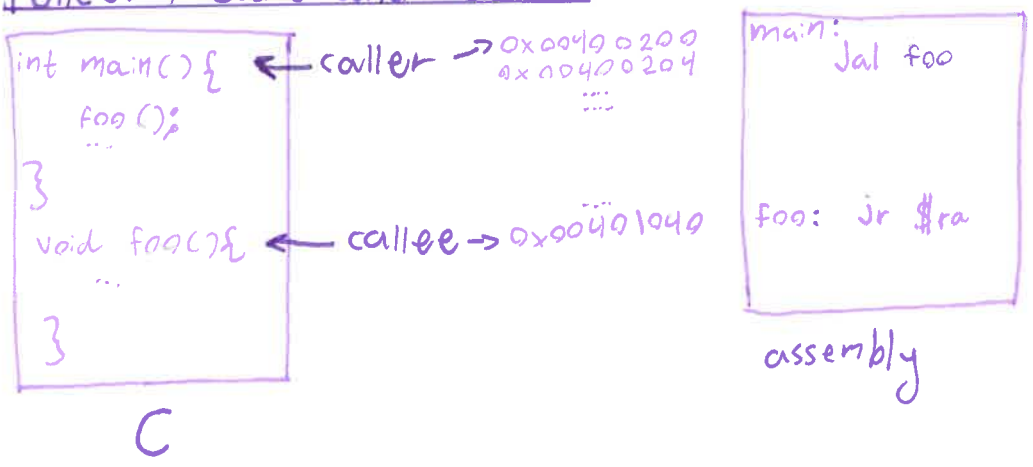
Itype Instruction:



J-Type Instruction:



function calls and return



Jal - Jump and link
↳ stores address of the next instruction in $ra
↳ Jumps to targetted address with program counter. $pc
Jr - Jump registers
↳ returns to the register of the next instruction.

Branch Delay Slot

When Jal is used, The next Instruction is executed before the Jump is done. Hence a nop (no operation) should be used to ensure there are no issues. However this can be used to make the code more efficient.

Globals

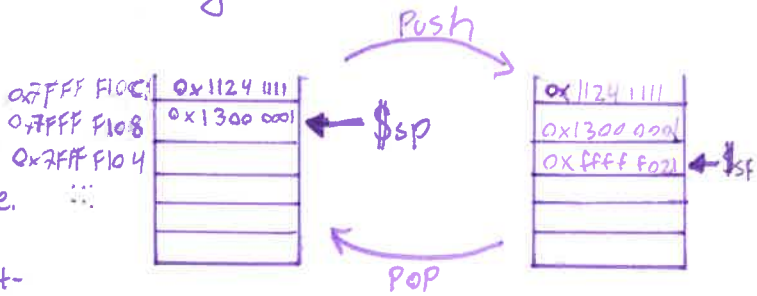
to make a function global, write ".global foo"

Arguments and return values

Arguments of a function are meant to be stored in $a0-$a4, and return values are meant to be stored in $v0, $v1. This means the arguments and return values are limited to 4 and 2 respectively.

The Stack

The Stack is a datastructure that is used to store registers into memory to allow more space.



The Stack follows a LiFo structure. elements are pushed and popped from the top of the stack. It uses the $sp (stack pointer) register to keep track of the top.

```
.macro PUSH (%reg)
    addi $sp, $sp, -4
    sw %reg, 0($sp)
.end_macro

.macro POP (%reg)
    lw %reg, 0($sp)
    addi $sp, $sp, 4
.end_macro
```

```
PUSH ($s0)
PUSH ($s1)
... reverse
POP ($s1)
POP ($s0)
```

Lecture 4

Primitive Datatypes

Name	Size (bits)	Min	Max
Char	8	-2^7	2^7-1
Unsigned char	8	0	2^8-1
Short	16	-2^{15}	$2^{15}-1$
Unsigned short	16	0	$2^{16}-1$
long	32/64		
int	Machine dependent		
Unsigned int	Machine dependent		
float	32		
double	64		

Floating point numbers

represented with 32 (float) or 64 (double) bits. Consist of 3 parts:

$3.7 \times 10^{-3} = 0,0037$

Mantissa, Base, exponent

Special values: +infinity, -infinity, NaN (not a number)

Do & While loops in C

```
int x = 0;
while (x<10){
    x++;
    Print("%d\n", x);
}
```

```
int x = 0
do{
    x++;
    Print(%d\n);
} while (x<10);
```

Both print 1-10

However if x was set to 10 at the start the while loop would not execute, but the do/while does.

Switch & Case

```
int x = 2;
switch (x){
    Case 1:
        Printf("1");
        break;
    case 2:
        printf("2");
        break;
    default:
        printf("none");
}
```

Will Print "2"

Functions in C

```
int sum(int x, int y){
    return (x+y);
}
```

(y: argument; (x+y): return value)

values declared inside a function are not accessible outside the function

Arrays in C

```
int b[3];
b[0] = 1;
b[1] = 2;
b[2] = 3;
```

← same thing

```
int b[] = {1, 2, 3};
```

The length is not needed when since it can be implied.

```
int len = sizeof(b)/sizeof(int)
```

number of entries in array; size of whole array in bytes; size of single entry

Pointers in C

```
int* x;      ← declare an int pointer
int  y=5;    ← declare an int =5
x = &y;      ← Point x to the address of y
*x = 8;      ← Set value at address to 8
Printf(y);
y = 8        ← y now has value 8
```

```
int x = b[2];
```

sets x to the value of the third element.

(0, 1, 2)

Malloc & free in C

Malloc allows for dynamic memory allocation. So that memory can be made after the code is compiled.

```
int* buf = (int*) malloc (sizeof(int)*n)
...
free (buf);
```

Malloc memory must be freed

I/O Devices

I/O or pheripherals are in/out-put devices.

Device	Behaviour	Rate
Keyboard	I	100 bit/s
Mouse	I	3.8 Kbit/s
Printer	O	3.2 mbit/s
Graphics	O	256 Gbit/s
Lan	I/O	10-100 Gbit/s
Memory	I/O	32-200 Mbit/s

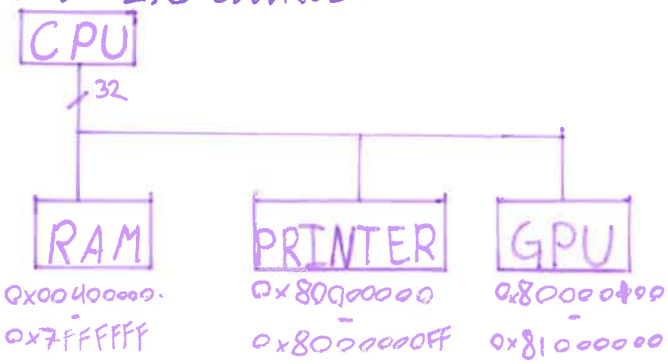
Volatile c keyword

Volatile is a compiler time keyword for variables to ensure that the program looks up the value every time to ensure that it did not change. IE to check the state of a button.

```
Volatile int* toggle18 = ....
```

I/O in Chipkits

The Chipkits use memory address of 0x80000000+ to communicate with I/O devices.

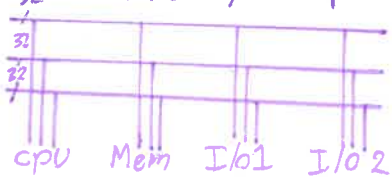


So the processor treats the I/O as additional memory address.
(x86 architectures are more complex)

The pins on the chip kit has two values: TRIS$_x$ - 0- Output 1-input
PORT$_x$ - 1-output 1
0-Output 0

Buses & DMA

A BUS is the inteface with which the cpu communicates with the pheripherals with.



Direct Memory Access (DMA)

When transferring data The CPU would be very busy if every transfer needed to go throught it. Instead a dma module is used to transfer data direcly between The memory and I/O devices to save the CPU some work.

Synchronous vs Asynchronus

Synchronus Busses have an additional clock wire. All operations use the clock to read from the Bus

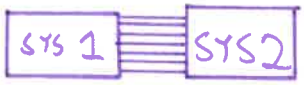
Asynchronus Busses do not have a clock but instead use a handshake to communicate without Interference.

Lecture 6

Parallell vs Serial Communication

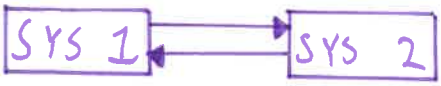
Parallell:

Several vires that communicate in parallen.



Serial:

Single wire that works in sequence.



examples: Spi, I^2C, UART

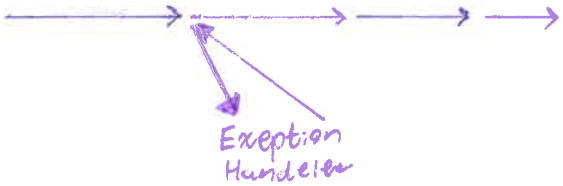
serial peripheal interface

Inter-integrated circuit

Exeptions & Interrupts

Exeptions are intennal erras that must be handeled by the soft or hand ware.

Interrupts are external to a program and can be issued by the system or by extenal sources such as a keyboard.



Error handeling in MIPS

Exeption handeler is located at 0x80000180

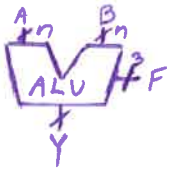
1) store current PC in $EPC
2) store clause of exeption in register called $clause
3) Jump to exeption handeler (0x80000180)
4) exeption handeler saves registers on stack
5) exeptin handeler inspets exeption and handles exeption.
6) exeption handeler restores registers and jumps back to $EPC. Using $k registers

And then a bunch of stuff about the mini-project.

Lecture 9 (Lecture 7 & 8 is is1500)

Arithmetic Logic Unit (ALU)

An alu is a hardware component that does arithmetic operations.

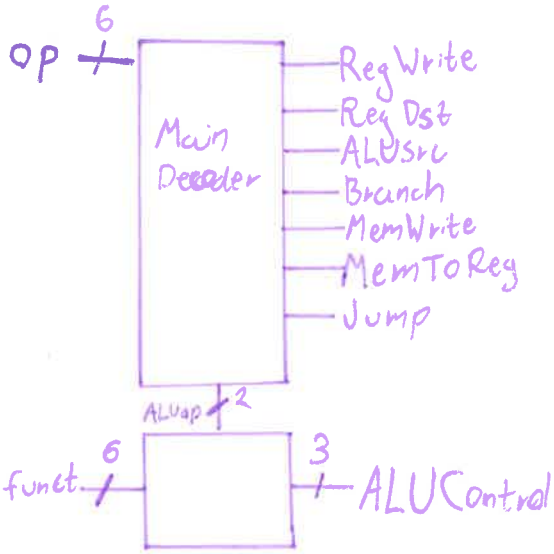


Alu also has Outputflags
Overflow - adder overflow
Zero - Answer is zero
Negative - Value is negative
Carry - result of addition

$F_{2:0}$	Function
000	A & B
001	A | B
010	A + B
011	
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Single-cycle Processor

I wont draw it, too hard. see Lecture slides.
The Sc Processor has flags to read different type of functions. They are set using the MIPS op code.



Preformance Analysis

Execution time is a Way of Determening perfor mence

$$\text{execution time (seconds)} = \#\text{instructions} \times \frac{\text{clk cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{CLK cykle}}$$

- #instructions ↑ Instructions in program
- clk cycles / instruction ↑ average cykles per instruction (CPI)
  - for all single cykel processors is 1
- seconds / CLK cykle ↑ CLK period T_c

Lecture 10

Parallelism & Pipelining

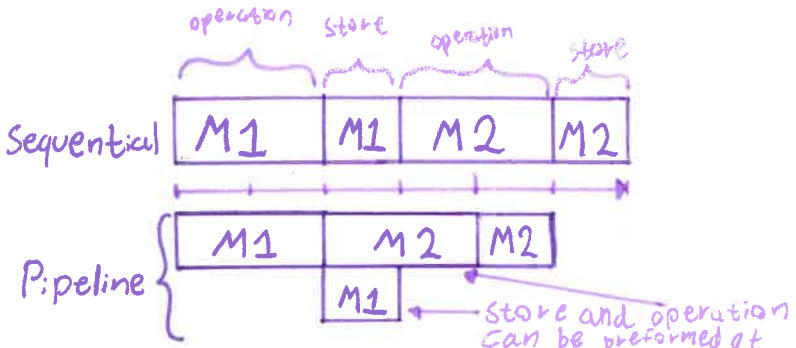
Definitions:

Processing system - A system that takes inputs and produces output

Token - An input that is being Processed into an output.

Latency - Delay it takes to process a single Token.

Throughput - number of tokens that can be processed per unit time.



Pipelining is Temporal parallelism.
parallell processing is spacial parallelism.

Performance Analysis

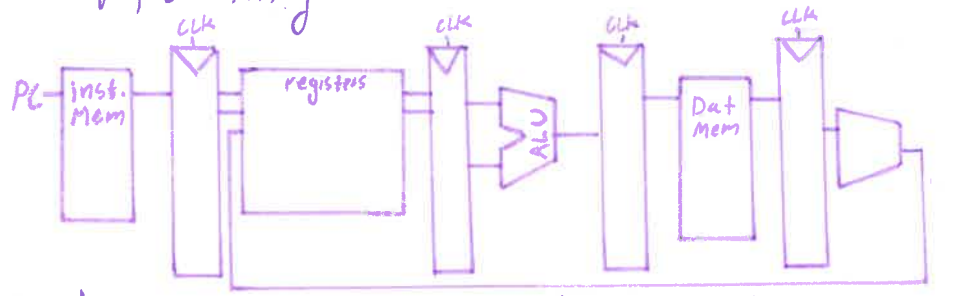
$$\text{Execution Time (seconds)} = \#\text{instructions} \times \frac{\text{clk cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

Pipelining makes the CPI slightly worse.

T_c^{-1} is greatly increased with pipelines

Pipelined Datapath

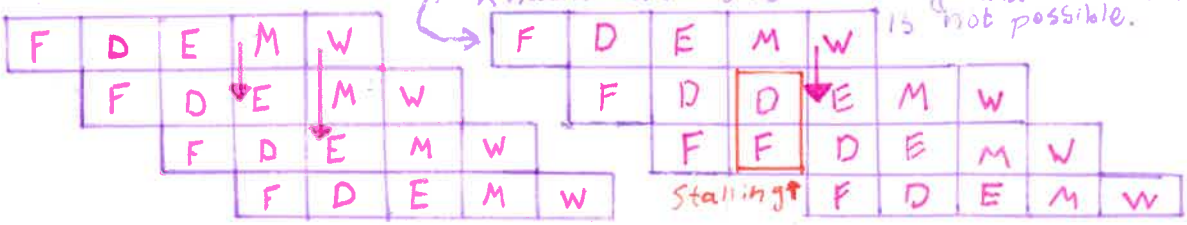
Splitting a single cycle in to 5 smaller cycles achives a higher frequency and allows for pipelining



Datahazard (Read after write)

In pipeline, it can happen that consequtive lines execute before a writeback.

Add $s0, $s1 $s2
sub $t0, $s0, $t0
and $t2, $t1, $s0
xori $t3, $s0, 12



if first instruction was lw $s0 20($s2)
but the rest are the same,
Then the future content of $s2 are only known at the M stage and forwarding is not possible.

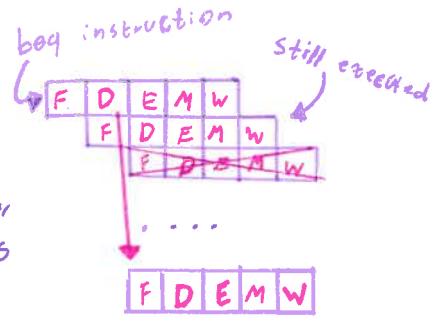
$s0 is not written to in time for sub to use it
This can be solved by adding headware that forwards from M and W to the exec state.

Last case scenario the processor stalls until the needed data can be forwarded

Control Hazard

When Jump instructions are executed the next instructions are not known. So they need to be "flushed" out. In a 5 stage pipeline this would mean 4 cycles would go to waste.

By executing the branch address calculation in the Decode stage so it is known after 1 cycle where to fetch the next instruction, resulting in that only one single cycle is wasted, or with delayed branching can be implemented efficiently.



At the decode stage you know the branch will be taken, hence the next Instructions will be fetched in the next cycle. Meaning only one line will be executed after the beq before the Jump.

Adding more pipelining

Pros:
+ More stages decreases the Critical path of a cycle
+ Clock frequency can be increased
+ Intel 2duo has 10 pipeline st

Cons:
- Adds hardware (registers)
- increases logic
- branch misprediction penalty

Branch Prediction

Static:
The compiler predicts if branch will be taken.

Dynamic:
Hardware predicts at the fetch state using a table of branch instructions.

ARMv7

- Used in embedded devices.
- More complex addressing than MIPS
- Uses ALU flags
- 16 gp registers
- 32 bit instructions

x86

- Used in Desktops & laptops
- very complex instruction set
- 8 gp registers
- Variable instruction length.

Lecture 11

Memory Technologies

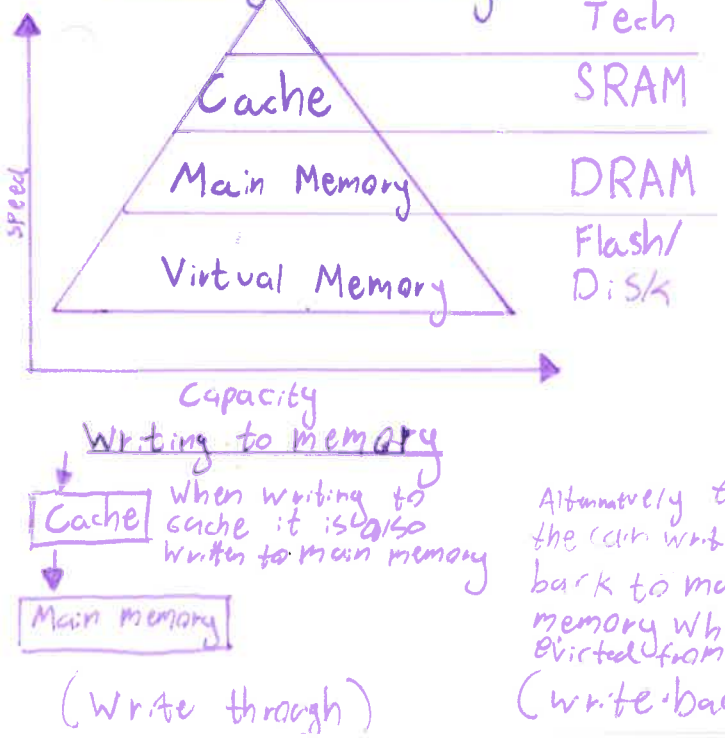
SRAM
- Static RAM
- Simple IC
- 0.5-2.5 ns
- 1000$/GiB
- Volatile

DRAM
- Stored in capacitor
- One transistor per bit.
- 50-70 ns
- 10-20 $/GiB

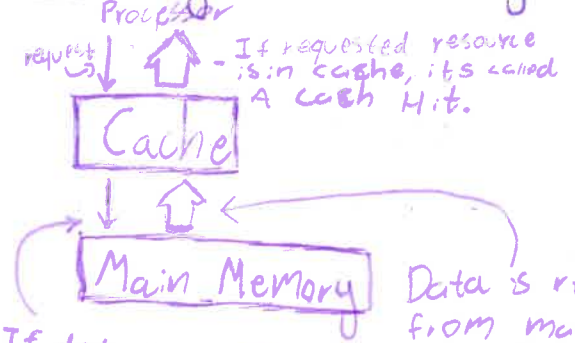
Flash
- Erasable read-only memory
- wears out
- Used in ssd.
- 5000-50000 ns
- 1$/Gib

Disk
- Spinning disk 5400-15000 rpm
- 5000000-50000000 ns
- 0,1$/Gib

Memory Hierarchy



Reading from Memory



- If requested resource is in cache, its called A cash Hit.

If data is not found in Cache it is a Cache miss and a request is sent

Data is returned from main memory and stored in cache. replacing old things.

Writing to memory

When writing to cache it is also written to main memory

(Write through)

Alternatively the cash writes back to main memory when evicted from cash

(write-back)

Virtual memory

Virtual memory uses the mass storage device (HDD/SSD) to store data. This is very slow but ensures capacity & gives memory protection.

Decreasing Missrate

Temporal Locality:
- Recently used data will likely be used again.

Spacial Locality:
- Data located near each other are often used together.

Both spacial and temporal locality are used together to predict what to store in the cache and reduce miss rates.

The Cache

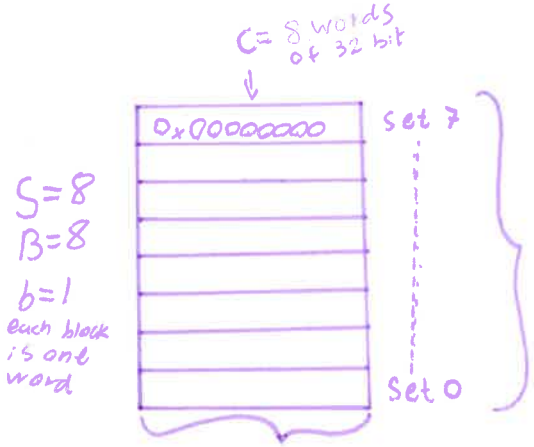
Capacity (C):
Number of words/bytes.

Set (S):
Number of blocks that held by a set.

Block size (b):
Number of words/bytes in a block.

Number of Blocks (B):
Total number blocks.
$B >= S$

Degree of Associativity:
$N = \frac{B}{S}$



N=1 => Direct map
N>1 => fully associative

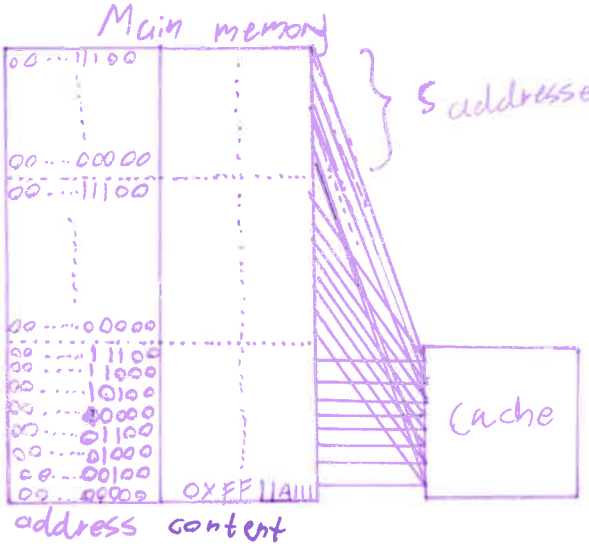
S=8

1×8=8=C

b=1 since each each each block is 1 word

Direct Mapped Cache

The main memory is mapped onto the cache based on position. Many overlaps occur.



Address components (field)

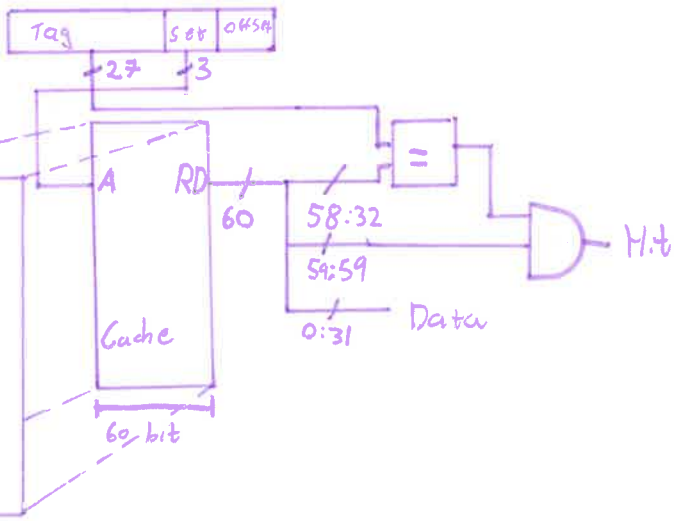
tag field

set/index field

Addresses which set in the cache so length in bits is $\log_2(S)$

Addresses each byte in block so the length is $\log_2(b)$

Specifies the address in main memory. The tag size is the remaining bits.



1 more bit is used to make sure the cache entry is valid

Moore's Law

Integrated Circuit transistor dencity doubles every 18-24 months

- Gordon Moore Intel co-founder 1960as

Dennard's Scaling:
More transistors means higer power consumption, resulting in lower voltage being used

Power wall

Since 2006 improvements are made through multiprocessing since increasing clockspeeds implies higher power consumption.
Current processors are limited by power draw.

Multiprocessor

A System With two or more processors.

Multicore- Many processors (cores) on a Single integrated circuit
Cluster- Computers in a LAN that can be viewed as a Single processor.

Performance- Tasks can be done in parallell
Energy- Many efficient small processors
Dependability- if 1 out of N processors fail n-1 other can continue.

Concurrency & Parallellism

Concurrency is handeling many things at once. (software view)
Parallellism is executing many things at once. (Hardware view)

Speedup

How much can performance be improved by parallelization?

$$\text{Speedup} = \frac{T_{before}}{T_{after}}$$

Amdahls Law

Speedup cannot always be linear.

$$T = T_{affected} + T_{unaffected}$$

$$T_{after} = \frac{T_{affected}}{N} + T_{unaffected}$$

(N: n cores)

$$\Rightarrow \text{Speedup} = \frac{T_{before}}{\frac{T_{affected}}{N} + T_{unaffected}}$$ } Amdahls law

Classes of parallelism

Data-Level

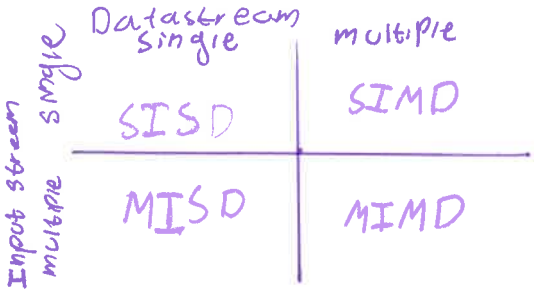
↳ Many items can be processed at the same time. For example Shearing multiple sheeps at once

Task-Level

↳ Many different tasks can be done in parallell. For example different tasks on a farm

SISD, SIMD, MIMD

To do with Instruction set and with data Stream.
Old classification (1960)



Instruction level parallelism

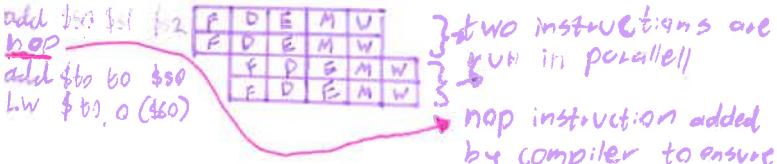
Increase performance without the involvement of the programmer. Uses SISD, SIMD or MIMD.
Two approaches:
1. Deeper pipelines
2. Multiple issues (many instructions per cycle)

Multiple Issue Approach

Many instructions per cycle. But gives problems since instructions usually rely on each other. Two solutions:

Static Multiple Issue:

↳ compiler ensures instructions are sane, and don't cause Hazards.



This uses very long instruction word (VLIW) that are 64 bits instead of 32 so that an instruction contains two instructions.
This means some additional hardware is needed in the pipeline.
↳ +1 ALU, +2 register read ports, +1 register write ports, +32 bit instruction memory.

Observations:

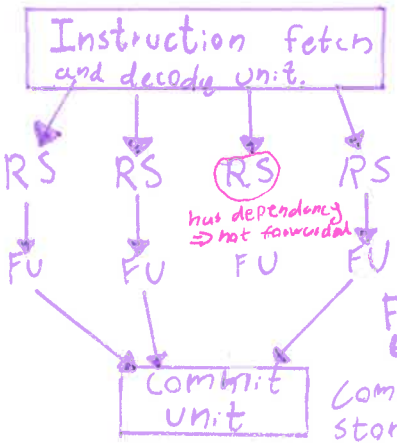
↳ VLIW is more energy efficient because less hardware is used.

↳ Superscalar can hide certain latencies

↳ Morden processors can issue up to 6 instructions per cycle but is often limited to 2 ipc due to dependencies

Dynamic Multiple Issue:

↳ instruction order is determined by hardware in the processor during execution.
Also called superscalar processing



Reservation Station
Takes care of dependency and forwards instructions without dependancy

Function unit executes instructions

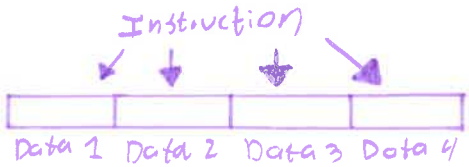
Commit unit stores results back in the registers.

Read after write (RAW)
The pipeline must ensure that when reading the latest value is given

Write after Read (WAR)
to avoid out of order execution writing before the read, writes are changed to special registers.

Lecture 13

Subword parallelism and multimedia extentions



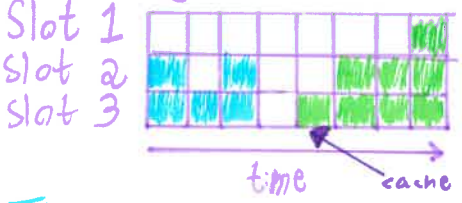
The same instruction is applied to every data.

- Wide dataword operated in parallell (same as SIMD)
- includes:
  - MMX multimedia extention Intel 1997
  - 3D Now! float operation AMD 1999
  - SSE/2 float operation Intel 1999
  - AVX (advanced vector extention) Intel & AMD 2011
  - NEON (multimedia ARMv7/v8
  - SVE Specialized scalable vector extention ARMv8

Hardware Multithreading

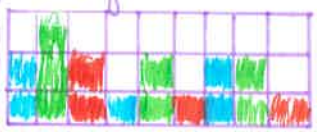
- several threads share FU (function unit, see lecture 12)

Coarse grained M. T.



Thread 1 switches to Thread 2 when a cache miss occurs to save on latency that a fetch from main memory would take.

Fine grained multithreading



Thread 1, 2, 3 are switched between every cycle, preventing stalls caused by dependencies or by cache misses

Simultaneous multithreading

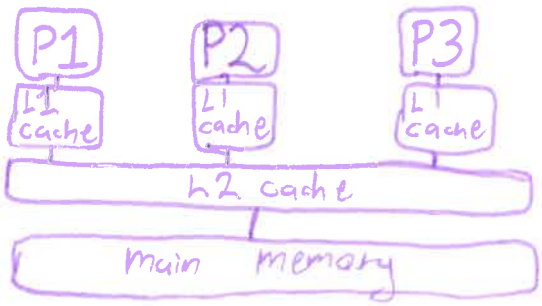
Threads 1, 2, 3 share the slots so the utilization is a lot higher. Also known as hyperthreading

GPU

- MIMD & SIMD & ILP
- CUDA.
- Parallell processing with CUDA
- Single instruction/multiple threads
- multithreaded SIMD
- Hide latency with multithreading

Shared Memory Multiprocessor

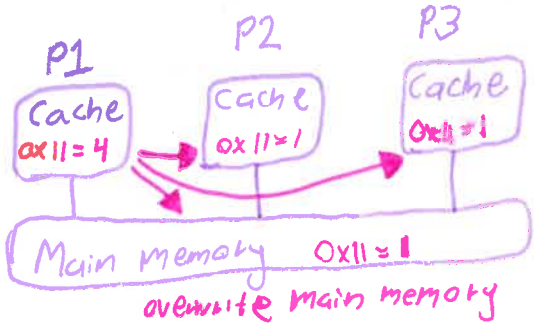
- Multiple processes share Main Memory
- Usually same as Multicore processor
- allows threads to communicate
- Creates problem with cache coherence



Snooping Protocol

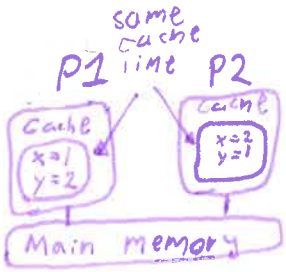
When a processor writes to it's cache and subsequentially to the main memory, it must invalidate one address for all other caches to make them fetch the new value. This solves the cache coherence problem

invalidate P2 & P3 caches



False Sharing

When two processes are working on the same cache line but different values, the line will be invalidated for both processes despite not needing to.



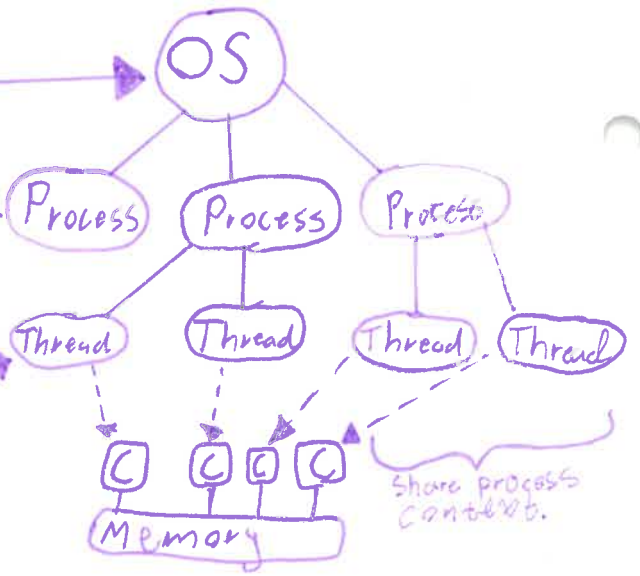
P1 Modifies X
P2 Modifies Y
both lines will be invalidated.

Processes, Threads & Cores

A modern OS can execute and switch between different processes at the same time.

A process context include a virtual memory space, IO/files, Read only code, heap, shared library, Process Id etc.

Each process has N number of threads the context includes thread ID, stack, stack pointer, program counter etc...
They are scheduled by the OS To execute in parallel on different cores.



Sharing variables (Mutex & semaphores)

When sharing variables between threads the programmer must ensure that the variable is not written to by two threads at the same time

Semaphore: (s)
Variable that can take postive values.
if s=0 then you cannot read or write to the protected area.

```
P(s) {
  while(s==0){ wait until s=0
    ;
  }
 *s -= 1;
}
V(s) {
 *s += 1        add +1 to s
}
```

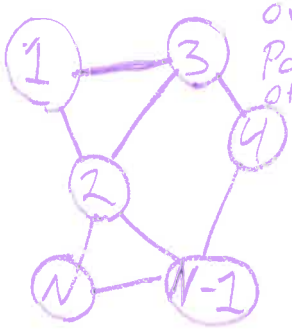
```
Semaphore s = 1;

P(s); ← Locks mutex
/*
  protected code } mutex
*/
V(s); ← unlocks mutex
```

Clusters & Warehouse Scale Computers

Set of computers connected over LAN
100 000 computers work together as one.
MapReduce Model:

1. Map:
   Applies programmer defined function to all data items.
2. Reduce:
   Collects outputs and applies another programmer defined function.



Communicate over message Passing instead of shared memory.

Supercomputers

Based on performance instead of paralellization. Measured in teraflops which is the calculations of doubles.