Lecture 3, Performance

```
Repeating some definitions:

  CPI    Clocks Per Instruction

  MHz    megahertz, millions of cycles per second

  MIPS   Millions of Instructions Per Second = MHz / CPI

  MOPS   Millions of Operations Per Second

  MFLOPS Millions of Floating point Operations Per Second

  MIOPS  Millions of Integer Operations Per Second

  (Classical, old, terms. Today would be billions.)


  Amdahl's Law (many forms, understand the concept)

           the part of time improved

new time = -------------------------  +  the part of time not improved

             factor improved by

old time = the part of time improved + the part of time not improved

          old time

speedup = --------     (always bigger over smaller when faster)

          new time

Given: on some program, the CPU takes 9 sec and the disk I/O takes 1
sec

What is the speedup using a CPU 9 times faster?

                   9 sec

Answer: new time = ----- + 1 sec = 2 sec
```

```
                   9


          old time = 9 + 1 = 10 sec

          speedup = 10 / 2 = 5   a pure number


----------------------------------------------------------------------

  Amdahl's Law (many forms, understand the concept)

             new performance

   speedup = ---------------

             old performance


   Given: Performance of M1 is 100 MFLOPS and 200 MIOPS

          Performance of M2 is  50 MFLOPS and 250 MIOPS

          On a program using 10% floating point and 90% integer

          Which is faster?  What is the speedup?

   Answer; .1 * 100 + .9 * 200 = 190 MIPS

           .1 *  50 + .9 * 250 = 230 MIPS   (M2 is faster)

           speedup = 230/190 = 1.21

----------------------------------------------------------------------


                        old performance

new performance = ----------------------------------------------------

                  fraction of old improved
```

```
                      ----------------------- + fraction of old unimproved

                         improvement factor



   Given: half of a 100 MIPS machine is speeded up by a factor of 3
What is the speedup relative to the original machine?

                             1                1

Answer: new performance = ---- * 100 MIPS = --- * 100 MIPS = 150 MIPS

                            0.5              .666

                            --- + 0.5

                             3

                                                 1

speedup = 150 / 100 = 1.5 (same as ----------------------------)

                                   fraction improved

                                   ---------    + fraction unimproved

                                   improvement factor

speedup is a pure number, no units. The units must cancel.


----------------------------------------------------------------------
```

SPEC Benchmarks

The benchmarks change infrequently, for example 2006 - 2016 are the same.

The speed seems to increase every year.

SPEC Int2006, 9 in C, 3 in C++

SPEC Flt2006, 17 in assorted Fortran, C, C++

SPEC many rules to follow

recent results

Note number of core available, results seem to be using just one core.

```
----------------------------------------------------------------------
```

CPI is average Clocks Per Instruction. units: clock/inst

MHz is frequency, we use millions of clocks per second. units: clock/sec

MIPS is millions of instruction per second. units: inst/sec

Note: MIPS=MHz/CPI because (clock/sec) / (clock/inst) = 10^6 inst/sec

(5/4 of people do not understand fractions.)

```
                  Computing average CPI, Clocks Per Instruction

     -------given---------------    ----------compute------------


     type  clocks  %use             product


     RR      3      25%             3 * 25 =  75

     RM      4      50%             4 * 50 = 200

     MM      5      25%             5 * 25 = 125

                  ______                    ____
```

```
              100%                      400    sum

                             400/100 = 4 average CPI



-------given---------------    ----------compute------------
type  clocks  instructions     product
RR      3       25,000         3 * 25,000 =  75,000
RM      4       50,000         4 * 50,000 = 200,000
MM      5       25,000         5 * 25,000 = 125,000
               _______                      _______
               100,000                      400,000     sum

                               400,000/100,000 = 4 average CPI


Find the faster sequence of instructions  Prog1 vs Prog2
-------given---------------------
type    clocks
 A        1
 B        2
 C        3
instruction counts for  A   B   C
Prog1                   2   1   2
```

```
Prog2                   4   1   1




----------compute------------------------------

Prog1

A    1   2     1 * 2 = 2

B    2   1     2 * 1 = 2

C    3   2     3 * 2 = 6

                      __ sum

                      10 clocks


Prog2

A    1   4     1 * 4 = 4

B    2   1     2 * 1 = 2

C    3   1     3 * 1 = 3

                      __ sum

                       9 clocks   more instructions yet faster

speedup = 10 clocks / 9 clocks = 1.111   a number (no units)
```

```
cs411_opcodes.txt different from Computer Organization and Design
2/7/16

rd is register destination, the result, general register 1 through 31

rs is the first register,  A, source, general register 0 through 31
```

```
rt is the second register, B, source, general register 0 through 31


--val---- generally a 16 bit number that gets sign extended

--adr---- a 16 bit address, gets sign extended and added to (rx)

"i" is generally immediate, operand value is in the instruction

Opcode Operands    Machine code format

                       6   5   5   5   5   6  number of bits in field

nop                RR  00  0   0   0   0   00

add    rd,rs,rt    RR  00  rs  rt  rd  0   32

sub    rd,rs,rt    RR  00  rs  rt  rd  0   34

mul    rd,rs,rt    RR  00  rs  rt  rd  0   27

div    rd,rs,rt    RR  00  rs  rt  rd  0   24

and    rd,rs,rt    RR  00  rs  rt  rd  0   10

or     rd,rs,rt    RR  00  rs  rt  rd  0   13

srl    rd,rt,shf   RR  00  0   rt  rd  shf 03

sll    rd,rt,shf   RR  00  0   rt  rd  shf 02

cmpl   rd,rt       RR  00  0   rt  rd  0   11

j      jadr        J   02  ------jadr--------

addi   rd,rs,val   M   07  rs  rd  ---val----

beq    rs,rt,adr   M   31  rs  rt  ---adr----

lw     rd,adr(rx)  M   35  rx  rd  ---adr----

sw     rt,adr(rx)  M   43  rx  rt  ---adr----
```

```
        instruction bits (binary of 6 5 5 5 5 6 format above)

    3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1

    1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0

               |         |         |         |         |

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 nop



    0 0 0 0 0 0 a a a a a b b b b b r r r r r -ignored- 1 0 0 0 0 0 add
    r,a,b

    0 0 0 0 0 0 a a a a a b b b b b r r r r r -ignored- 1 0 0 0 1 0 sub
    r,a,b

    0 0 0 0 0 0 a a a a a b b b b b r r r r r -ignored- 0 1 1 0 1 1 mul
    r,a,b

    0 0 0 0 0 0 a a a a a b b b b b r r r r r -ignored- 0 1 1 0 0 0 div
    r,a,b

    0 0 0 0 0 0 a a a a a b b b b b r r r r r -ignored- 0 0 1 0 1 0 and
    r,a,b

    0 0 0 0 0 0 a a a a a b b b b b r r r r r -ignored- 0 0 1 1 0 1 or
    r,a,b

    0 0 0 0 0 0 0 0 0 0 0 b b b b b r r r r r s s s s s 0 0 0 0 1 1 srl
    r,b,s

    0 0 0 0 0 0 0 0 0 0 0 b b b b b r r r r r s s s s s 0 0 0 0 1 0 sll
    r,b,s

    0 0 0 0 0 0 0 0 0 0 0 b b b b b r r r r r -ignored- 0 0 1 0 1 1 cmpl
    r,b

    0 0 0 0 1 0 -----address to bits (27:2) of PC------------------ j adr

    0 0 0 1 1 1 x x x x x r r r r r ---2's complement value-------- addi
    r,val(x)

    0 1 1 1 1 1 a a a a a b b b b b ---2's complement address------ beq
    a,b,adr
```

```
1 0 0 0 1 1 x x x x x r r r r r ---2's complement address------ lw
r,adr(x)

1 0 1 0 1 1 x x x x x b b b b b ---2's complement address------ sw
b,adr(x)
```

Definitions:

nop   no operation, no programmer visible registers or memory are changed, except PC gets PC+4

j adr   bits 0 through 25 of the instruction are inserted into PC(27:2) probably should zero bits PC(1:0) but should be zero already

lw r,adr(x) load word into register r from memory location (register x plus sign extended adr field)

sw b,adr(x) store word from register b into memory location (register x plus sign extended adr field)

beq a,b,adr branch on equal, if the contents of register a are equal to the contents of register b, add the, shifted by two, sign extended adr to the PC (The PC will have 4 added by then)

addi r,val(x) add immediate, the contents of register x is added to the sign extended value and the result put into register r

add r,a,b   add register a to register b and put result into register r

sub r,a,b   subtract register b from register a and put result into register r

mul r,a,b   multiply register a by register b and put result into regster r

div r,a,b   divide register a by register b and put result into register r

and r,a,b   and register a to register b and put result into register r

or r,a,b   or register a to register b and put result into register r

srl r,b,s   shift the contents of register b by s places right and put result in register r

sll r,b,s   shift the contents of register b by s places left and put result in register r

cmpl r,b   one's complement of register b goes into register r

Also: no instructions are to have side effects or additional "features"

General register list (applies to MIPS ISA and project)

(note: project op codes may differ from MIPS/SGI)

Register			notes
0	$0		zero value, not writable
1	$1		
2	$2	$v0	return values (convention, not constrained by hardware)
3	$3	$v1	
4	$4	$a0	arguments (convention, not constrained by hardware)
5	$5	$a1	
6	$6	$a2	
7	$7	$a3	
8	$8	$t0	temporaries(not saved by software convention over calls)
9	$9	$t1	
10	$10	$t2	

```
11   $11  $t3

12   $12  $t4

13   $13  $t5

14   $14  $t6

15   $15  $t7

16   $16  $s0     saved by software convention over calls

17   $17  $s1

18   $18  $s2

19   $19  $s3

20   $20  $s4

21   $21  $s5

22   $22  $s6

23   $23  $s7

24   $24  $t8     more temporaries

25   $25  $t9

26   $26

27   $27

28   $28  $gp     global pointer ( more designations by software
convention)

29   $29  $sp     stack pointer

30   $30  $fp     frame pointer

31   $31  $ra     return address


Remember: From a hardware view registers 1 through 31 are general
```

purpose and identical. The above table is just software conventions.

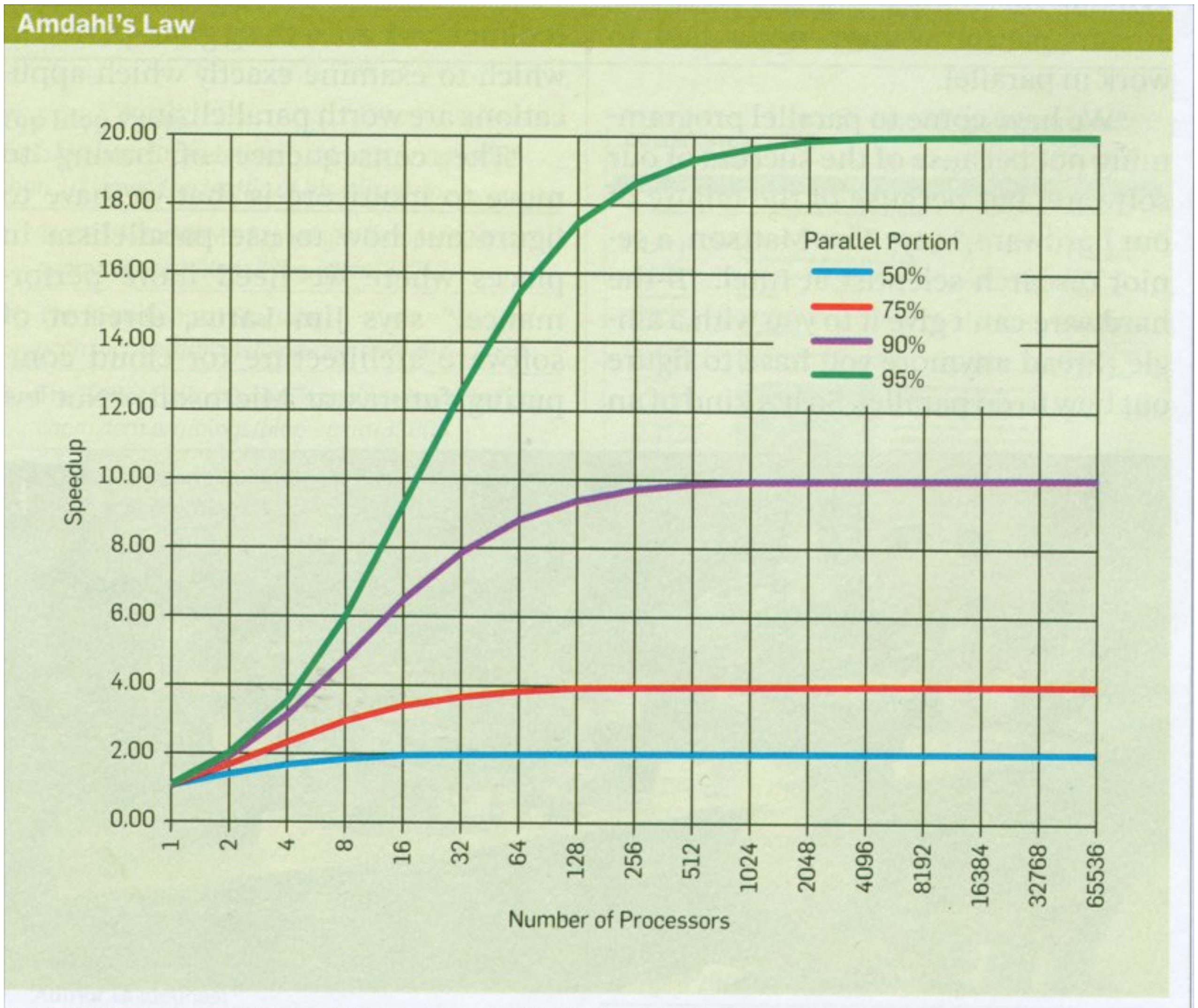
Register zero is always zero!

Basic digital logic

IA-64 Itanium

We will cover multicore and parallel processors later.

Amdahls law applies to them also.



Named after computer architect Gene Amdahl, Amdahl's Law is frequently used in parallel programming to predict the theoretical maximum speedup using multiple processors.