Disclaimer

The slides presented here are a combination of the CS251 course notes from previous terms, the work of Xiao-Bo Li, and material from the required textbook "Computer Organization and Design, ARM Edition," by David A. Patterson and John L. Hennessy. It is being used here with explicit permission from the authors.

CS251 course policy requires students to delete all course files after the term. Therefore, please do not post these slides to any website or share them.

CS251 - Computer Organization and Design

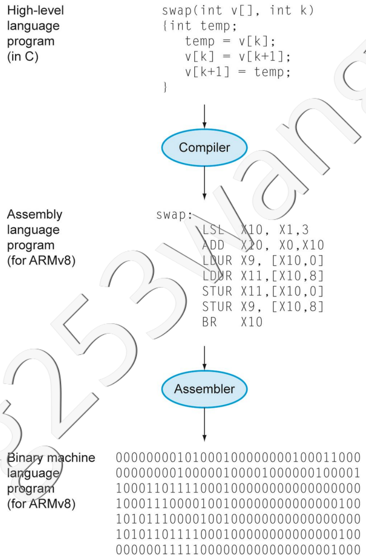
Performance - factors and measurement

Instructor: Zille Huma Kamal

University of Waterloo

Spring 2023

Review



```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

```
swap:
      LSL  X10, X1,3
      ADD  X20, X0,X10
      LDUR X9, [X10,0]
      LDUR X11,[X10,8]
      STUR X11,[X10,0]
      STUR X9, [X10,8]
      BR   X10
```

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

Figure 1.4 from the text

Textbook Chapter 1.1: Introduction - Understanding Program Performance

The performance of a program depends on:

- Algorithms (not this course) - Demo matrix element access
  Affect on performance: number of **source-level** statements, number of I/O operations
- Programming language, compiler, architecture - Demo `ARM vs x86`
  Affect on performance: number of **computer instructions** for each source-level statement
- Processor and memory - Demo `perf` utility command
  Affect on performance: how fast instructions are executed.
- IO system
  Affect on performance: how fast IO is executed.

Example: Small code changes, big performance differences

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      a[i][j]=32767; } } }
```

- Row-by-row (a[i][j]): 1.693 sec

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      a[j][i]=32767; } } }
```

- Down a column (a[j][i]): 27.045 sec (approx 16 times slower!)

Example on i7-2677M

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      a[i][j]=32767; } } }
```

- Row-by-row (a[i][j]): on a 32-bit i7 0.30 sec [0.056 sec on a 64-bit i7]
- By column (a[j][i]): on a 32-bit i7 1.24 sec [0.23 sec on a 64-bit i7]
  (approx 4 times slower!)

Example Revisited: Memory access vs. Registers

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  register int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      ; } } }
```

- `register int i,j`: 0.044 sec

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      ; } } }
```

- `int i,j`: 0.27 sec
  (approx 6 times slower!)

Performance: `perf` tool for beginners

- Demo `perf` to illustrate a system tool for gathering performance related metrics
- "improve" performance = increase performance.
- "improve" execution time = decrease execution time.
- The computer that performs the work in the least amount of **time** is the fastest. Nearly all computers have a clock.
  - Clock cycles
    Discrete intervals when hardware events take place.
  - Clock period: the length of a clock cycle.
  - Clock rate: 1 / clock period.
- What metric do we choose for time?
  - Wall clock time?
  - When did the job first start executing on the CPU?
  - Total time to complete a task including any disk or memory access?

CPU execution time `CPU time`

- When CPU is shared amongst processes, `CPU time` for a process A, is the time the CPU spends on process A, excluding time spent for I/O, or execution of other programs.
- Define what metrics are important and then find bottlenecks to improve performance.
- A metric for measuring performance: `CPU time`

  CPU time

  $$= \textbf{Number of } \text{(clock cycles) for a program} \times \textbf{length of } \text{(clock cycle)}$$

  $$= \text{Number of clock cycles for a program} \times \frac{1}{\text{clock rate}}$$

- To improve performance:
  - reduce the number of clock cycles for a program
  - reduce the length of the clock cycle
- There is a tradeoff between the two.

Cycles Per Instructions (CPI)

- CPI is another way of comparing two different implementations of the identical instruction set architecture.
- A program has a number of (machine/assembly) instructions, generated by the compiler.

$$\text{number of clock cycles} = \text{number of instructions} \times CPI$$

- CPI (clock cycles per instruction): the average number of clock cycles each instruction takes to execute.

Performance

- Other performance metrics include:
  - Response time: time between start and completion of a task
  - Throughput: total amount of work done in a given time
- Improving response time usually improves throughput
  - Replacing a slow processor with a faster one.
    - ★ Response time is faster, and so throughput has increased
  - Adding more processors, where each processor does ONLY one task.
    - ★ If no one task gets completes faster, then only throughput increases.

Example: Computing Performance

- Let us consider number of instructions executed as a performance metric.
- Consider the example available at: `/u/cs251/Simulator/arm/copy.arm`

Try this

How many instructions are executed?

```
000: ADDI X2,X31,#5
004: ADDI X3,X31,#104
008: ADDI X4,X31,#200
012: LDUR X1,[X3,#0]
016: STUR X1,[X4,#0]
020: ADDI X3,X3,#8
024: ADDI X4,X4,#8
028: SUBI X2,X2,#1
032: CBNZ X2,#-5
```

Computing execution time (number of clock cycles)

Try this

What is the execution time for the same set of instructions?

```
000: ADDI X2,X31,#5
004: ADDI X3,X31,#104
008: ADDI X4,X31,#200
012: LDUR X1,[X3,#0]
016: STUR X1,[X4,#0]
020: ADDI X3,X3,#8
024: ADDI X4,X4,#8
028: SUBI X2,X2,#1
032: CBNZ X2,#-5
```

If one instruction is executed per clock cycle?

If all instructions take 1 clock cycle, **BUT** CB-type instructions take 5 clock cycles?

Clock Cycle Example: Base Case

Suppose one instruction is executed per clock cycle.

```
0: ADDI X2,X31,#5         1
4: ADDI X3,X31,#104       1
8: ADDI X4,X31,#200       1
// code below repeated 5 times
12: LDUR X1,[X3,#0]       1
16: STUR X1,[X4,#0]       1
20: ADDI X3,X3,#8         1
24: ADDI X4,X4,#8         1
28: SUBI X2,X2,#1         1
32: CBNZ X2,#-5           1
```

Then, total clock cycle: $3 + 6 * 5 = 33$ cc

Clock Cycle Example: Branch Case

The following example assumes branching takes 5 cc, significantly more clock cycles than all other instructions.

```
0: ADDI X2,X31,#5         1
4: ADDI X3,X31,#104       1
8: ADDI X4,X31,#200       1
// code below repeated 5 times
12: LDUR X1,[X3,#0]       1
16: STUR X1,[X4,#0]       1
20: ADDI X3,X3,#8         1
24: ADDI X4,X4,#8         1
28: SUBI X2,X2,#1         1
32: CBNZ X2,#-5           5
```

total clock cycle: $3 + 10 * 5 = 53$ cc

Clock Cycle Example: Cache Motivation

Think About It

- If load and store takes 100 cc since RAM is slow, what would be the execution time for the `copy.arm` program?
- Can we avoid getting **data** from RAM? Yes, use a cache, it is much faster

Clock Cycle Example: Cache Motivation 1

If load and store takes 100 cc since RAM is slow, we have 1043 cc.

```
0: ADDI X2,X31,#5         1
4: ADDI X3,X31,#104       1
8: ADDI X4,X31,#200       1
// code below repeated 5 times
12: LDUR X1,[X3,#0]       100
16: STUR X1,[X4,#0]       100
20: ADDI X3,X3,#8         1
24: ADDI X4,X4,#8         1
28: SUBI X2,X2,#1         1
32: CBNZ X2,#-5           5
```

total clock cycle: $3 + 208 * 5 = 1043$ cc

Clock Cycle Example: Cache Motivation 2

Think About It

- Each instruction is in RAM. If RAM access is 100 cc, what would be the execution time for the copy.arm program?
- Can we avoid getting **instructions** from RAM? Yes, use a cache, it is much faster

```
0: ADDI X2,X31,#5           100
4: ADDI X3,X31,#104         100
8: ADDI X4,X31,#200         100
// code below repeated 5 times
12: LDUR X1,[X3,#0]         100 + 100
16: STUR X1,[X4,#0]         100 + 100
20: ADDI X3,X3,#8           100
24: ADDI X4,X4,#8           100
28: SUBI X2,X2,#1           100
32: CBNZ X2,#-5             100
```

Total clock cycle: $300 + (600 + 200) * 5 = 4300$ cc

Some factors affecting response time

- Speed of clock
- Complexity of instruction set
- Efficiency of compilers
- Mix of instructions needed to complete a task
- Some choices in designing computers:
  - simple instruction set, fast clock, one instruction executed per clock cycle
  - complex instruction set, faster clock, some instructions take multiple cycles to execute

Multicore Processors

- We have run into the power limit for cooling commodity microprocessors.
- Instead of decreasing the response time of one program on a single processor,
- Since 2006 all desktop and servers have microprocessors with multiple processors (cores) per chip.
- To get significant improvement in response time, programs must take advantage of **multiple cores**.

Benchmarks

- Standard set of programs (and data) chosen to measure performance
- Advantages:
  - Provides basis for meaningful comparisons
  - Design by committee may eliminate vendor bias
- Disadvantages:
  - Vendors can optimize for benchmark performance
  - Possible mismatch between benchmark and user needs
  - Still an artificial measurement

Textbook Readings

- Readings: skim Chapter 1, read section 1.6

Additional Slides

Remaining slides are additional notes for your information.

CS 251 vs CS 241

- CS 241 uses MIPS, CS 251 uses ARM

MIPS vs ARM

- MIPS and ARM assembly similar:

MIPS	ARM
`lw $1, 0($2)`	`LDUR X1, [X2,#0]`
`add $1,$2,$3`	`ADD X1,X2,X3`
`addi $1,$2,22`	`ADDI X1,X2,#22`

- ARMv7 has 15 registers; ARMv8 (LEG), MIPS have 32
  ARMv8,LEG: X31 is always 0
  MIPS: $0 is always 0
- ARM takes about 1/4 the transitors as MIPS
- ARMv7 has conditional forms of instructions
  ARM compiler takes advantage of this; gcc does not
  ARMv8 has fewer fancy features; we use none of them