HW ASSIGNMENTS:

Week 1 (Ch. 3 & 12.1 & 12.2 & 12.3):

3.3 Consider a hypothetical 32-bit microprocessor having 32-bit instructions composed of two fields: the first byte contains the opcode and the remainder the immediate operand or an operand address.

a. What is the maximum directly addressable memory capacity (in bytes)?
b. Discuss the impact on the system speed if the microprocessor bus has
   1. a 32-bit local address bus and a 16-bit local data bus, or
   2. a 16-bit local address bus and a 16-bit local data bus.
c. How many bits are needed for the program counter and the instruction register?

A:

a. 2^(32-8) = 2^24 = 16,777,216 bytes = 16 MB ,(8 bits = 1 byte for he opcode). **b.1**. a 32-bit local address bus and a 16-bit local data bus. Instruction and data transfers would take three bus cycles each, one for the address and two for the data. Since If the address bus is 32 bits, the whole address can be transferred to memory at once and decoded there; however, since the data bus is only 16 bits, it will require 2 bus cycles (accesses to memory) to fetch the 32-bit instruction or operand.

b.2. a 16-bit local address bus and a 16-bit local data bus. Instruction and data transfers would take four bus cycles each, two for the address and two for the data. Therefore, that will have the processor perform two transmissions in order to send to memory the whole 32-bit address; this will require more complex memory interface control to latch the two halves of the address before it performs an access to it. In
addition to this two-step address issue, since the data bus is also 16 bits, the microprocessor will need 2 bus cycles to fetch the 32-bit instruction or operand.

c. For the PC needs 24 bits (24-bit addresses), and for the IR needs 32 bits (32-bit addresses).

3.5 Consider a 32-bit microprocessor, with a 16-bit external data bus, driven by an 8-MHz input clock. Assume that this microprocessor has a bus cycle whose minimum duration equals four input clock cycles. What is the maximum data transfer rate across the bus that this microprocessor can sustain, in bytes/s? To increase its performance, would it be better to make its external data bus 32 bits or to double the external clock frequency supplied to the microprocessor? State any other assumptions you make, and explain. *Hint:* Determine the number of bytes that can be transferred per bus cycle.

A:

Remember the reciprocal relationship between frequency and period. Also it takes one bus cycle to transfer 2 bytes. How long is a bus cycle? See “tips log” for more tips.

Clock cycle = 1/(8 MHz) = $0.125*10^{-6}$ = 125 ns

Bus cycle = 4 * 125 ns = 500 ns

2 bytes transferred every 500 ns; thus transfer rate = 2/500ns = 4 MBytes/sec, where Mega is 10^6 in this case.

Doubling the frequency may mean adopting a new chip manufacturing technology (assuming each instructions will have the same number of clock cycles); doubling the external data bus means wider (maybe newer) on-chip data bus drivers/latches and modifications to the bus control logic. In the first case, the speed of the memory chips will also need to double (roughly) not to

slow down the microprocessor; in the second case, the "word length" of the memory will have to double to be able to send/receive 32-bit quantities.

Review Questions

12.1 What general roles are performed by processor registers?
12.2 What categories of data are commonly supported by user-visible registers?
12.3 What is the function of condition codes?
12.4 What is a program status word?

A:
12.1: User-visible registers: These enable the machine- or assembly language programmer to minimize main-memory references by optimizing use of registers. **Control and status registers:** These are used by the control unit to control the operation of the CPU and by privileged, operating system programs to control the execution of programs.
12.2: General purpose; Data; Address; Condition codes
12.3: Condition codes are bits set by the CPU hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero, or overflow result. In addition to the result itself being stored in a register or memory, a condition code is also set. The code may subsequently be tested as part of a conditional branch operation.
12.4: All CPU designs include a register or set of registers, often known as the *program status word* (PSW), that contain status information. The PSW typically contains condition codes plus other status information.

12.3 A microprocessor is clocked at a rate of 5 GHz.
a. How long is a clock cycle?
b. What is the duration of a particular type of machine instruction consisting of three clock cycles?

A:
(a)clock cycle = 1/(5*10^9) = 2*10^-10 sec
= 200 pico seconds
= 0.2 nano seconds
(b) 3*0.2 = 0.6 nano seconds

12.4 A microprocessor provides an instruction capable of moving a string of bytes from one area of memory to another. The fetching and initial decoding of the instruction takes 10 clock cycles. Thereafter, it takes 15 clock cycles to transfer each byte. The microprocessor is clocked at a rate of 10 GHz.
a. Determine the length of the instruction cycle for the case of a string of 64 bytes.
b. What is the worst-case delay for acknowledging an interrupt if the instruction is noninterruptible?
c. Repeat part (b) assuming the instruction can be interrupted at the beginning of each byte transfer.

A:
a.10+15*64=970 clock cycles=970/(10*10^9)=97 ns
b.10 clock cycles=1 ns
c.25 clock cycles=2.5 ns

Week 2 (Ch. 12.4):

12.5 Why is a two-stage instruction pipeline unlikely to cut the instruction cycle time in half, compared with the use of no pipeline?

12.6 List and briefly explain various ways in which an instruction pipeline can deal with conditional branch instructions.

12.7 How are history bits used for branch prediction?

A:

12.5

(1) The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some time before it can empty its buffer. (2) A conditional branch instruction makes the address of the next instruction to be fetched unknown. Thus, the fetch stage must wait until it receives the next instruction address from the execute stage. The execute stage may then have to wait while the next instruction is fetched.

12.6

Multiple streams: A brute-force approach is to replicate the initial portions of the pipeline and allow the pipeline to fetch both instructions, making use of two streams. Prefetch branch target: When a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch. This target is then saved until the branch instruction is executed. If the branch is taken, the target has already been prefetched. Loop buffer: A loop buffer is a small, very-high-speed memory maintained by the instruction fetch stage of the pipeline and containing the n most recently fetched instructions, in sequence. If a branch is to be taken, the hardware first checks whether the branch target is within the buffer. If so, the next instruction is fetched from the buffer. Branch prediction: A prediction is made whether a conditional branch will be taken when executed, and subsequent instructions are fetched accordingly. Delayed branch: It is possible to improve pipeline performance by automatically rearranging instructions within a program, so that branch instructions occur later than actually desired.

12.7

One or more bits that reflect the recent history of the instruction can be associated with each conditional branch instruction. These bits are referred to as a taken/not taken switch that directs the processor to make a particular decision the next time the instruction is encountered.

12.8 Assume a pipeline with four stages: fetch instruction (FI), decode instruction and calculate addresses (DA), fetch operand (FO), and execute (EX). Draw a diagram similar to Figures 12.10 for a sequence of 7 instructions, in which the third instruction is a branch that is taken and in which there are no data dependencies.

A:

Link:
http://www.chegg.com/homework-help/assume-pipeline-four-stages-fetch-instruction-fi-decode-inst-chapter-14-problem-8p-solution-9780134102061-exc

	1	2	3	4	5	6	7	8	9	10
Instruction 1	FI	DA	FO	EX						
Instruction 2		FI	DA	FO	EX					
Instruction 3			FI	DA	FO	EX				
Instruction 4				FI	DA	FO	EX			
Instruction 5					FI	DA	FO	EX		
Instruction 6						FI	DA	FO	EX	
Instruction 7							FI	DA	FO	EX

FI : Fetch Instruction

DA : Calulate address

FO : Fetch Operand

EX : Execute

12.9 A pipelined processor has a clock rate of 2.5 GHz and executes a program with 1.5 million instructions. The pipeline has five stages, and instructions are issued at a rate of one per clock cycle. Ignore penalties due to branch instructions and out-of-sequence executions.

a. What is the speedup of this processor for this program compared to a non-pipelined processor, making the same assumptions used in Section 12.4?
b. What is throughput (in MIPS) of the pipelined processor?

A:

a. What is the speedup of this processor for this program compared to a non-pipelined processor, making the same assumptions used in Section 14.4?
15 ms/3 ms = 5 times speedup

b. What is the throughput of the pipelined processor?
1.5 x 10^5 instructions/3.0 x 10 -3 sec = 500,000,000 instructions per second or 500,000 instructions per ms

---

Week 3 (Ch.13 <Lab 2>):

13.1 What are some typical distinguishing characteristics of RISC organization?

13.2 Briefly explain the two basic approaches used to minimize register-memory operations on RISC machines.

13.3 If a circular register buffer is used to handle local variables for nested procedures, describe two approaches for handling global variables.

13.4 What are some typical characteristics of a RISC instruction set architecture?

13.5 What is a delayed branch?

A:

13.1

(1) a limited instruction set with a fixed format, (2) a large number of registers or the use of a compiler that optimizes register usage, and (3) an emphasis on optimizing the instruction pipeline.

13.2

Two basic approaches are possible, one based on software and the other on hardware. The software approach is to rely on the compiler to maximize register usage. The compiler will attempt to allocate registers to those variables that will be used the most in a given time period. This approach requires the use of sophisticated program-analysis algorithms. The hardware approach is simply to use more registers so that more variables can be held in registers for longer periods of time.

13.3

(1) Variables declared as global in an HLL can be assigned memory locations by the compiler, and all machine instructions that reference these variables will use memory-reference operands. **(2)** Incorporate a set of global registers in the processor. These registers would be fixed in number and available to all procedures

13.4

One instruction per cycle. Register-to-register operations. Simple addressing modes. Simple instruction formats.

13.5

Delayed branch, a way of increasing the efficiency of the pipeline, makes use of a branch that does not take effect until after execution of the following instruction.

13.1 Considering the call-return pattern in Figure 4.21, how many overflows and underflows (each of which causes a register save/restore) will occur with a window size of
- **a.** 5?
- **b.** 8?
- **c.** 16?

A:
This one I need to figure out on my own

13.2 In the discussion of Figure 13.2, it was stated that only the first two portions of a window are saved or restored. Why is it not necessary to save the temporary registers?

A:
(see link)
http://www.chegg.com/homework-help/discussion-figure-stated-first-two-portions-window-saved-res-chapter-15-problem-2p-solution-9780132936330-exc

13.4 Reorganize the code sequence in Figure 13.6d to reduce the number of NOOPs.

A:
(NOOP = No Operation)
This one I need to figure out on my own…

Week 4 (Ch.14 <Lab 3>):

14.1 What is the essential characteristic of the superscalar approach to processor design?
14.2 What is the difference between the superscalar and superpipelined approaches?
14.3 What is instruction-level parallelism?
14.4 Briefly define the following terms:
- True data dependency
- Procedural dependency
- Resource conflicts
- Output dependency
- Antidependency

14.5 What is the distinction between instruction-level parallelism and machine parallelism?
14.6 List and briefly define three types of superscalar instruction issue policies.
14.7 What is the purpose of an instruction window?
14.8 What is register renaming and what is its purpose?
14.9 What are the key elements of a superscalar processor organization?

A:
14.1

Superscalar processor can execute multiple independent instructions in parallel. Within a single processor, it introduces a parallelism called "Instruction-Level Parallelism". Every processor has multiple execution units and this preprocessor dispatches multiple instructions to different execution units on the processor.

14.2

Superpipeline breaks the stages of a given pipeline into smaller stages. This is done by shortening the clock period for every instruction. Superpipeline is capable of performing 2 pipeline stages per clock cycle.

Superscalar doesn't decrease the clock cycle. Instead it depends on the processor's ability to execute multiple instructions in parallel. Superpipeline introduces a level of parallelism by increasing the number of instructions which can be in a pipeline.

14.3

Instruction-level parallelism (**ILP**) is a measure of how many of the **instructions** in a computer program can be executed simultaneously. There are two approaches to **instruction level parallelism**: Hardware. Software.

14.4

True data dependency: A second instruction needs data produced by the first instruction. **Procedural dependency:** The instructions following a branch (taken or not taken) have a procedural dependency on the branch and cannot be executed until the branch is executed. **Resource conflicts:** A resource conflict is a competition of two or more instructions for the same resource at the same time. **Output dependency:** Two instructions update the same register, so the later instruction must update later. **Antidependency:** A second instruction destroys a value that the first instruction uses.

14.5

Instruction-level parallelism exists when instructions in a sequence are independent and thus can be executed in parallel by overlapping. **Machine parallelism** is a measure of the ability of the processor to take advantage of instruction-level parallelism. Machine parallelism is determined by the number of instructions that can be fetched and executed at the same time (the number of parallel pipelines) and by the speed and sophistication of the mechanisms that the processor uses to find independent instructions.

14.6

In-order issue with in-order completion: Issue instructions in the exact order that would be achieved by sequential execution and to write results in that same order. **Inorder issue with out-of-order completion:** Issue instructions in the exact order that would be achieved by sequential execution but allow instructions to run to completion out of order. **Out-of-order issue with out-of-order completion:** The processor has a lookahead capability, allowing it to identify independent instructions that can be brought into the execute stage. Instructions are issued with little regard for their original program order. Instructions may also run to completion out of order.

14.7

Instruction window. An **instruction window** in computer architecture refers to the set of **instructions** which can execute out of order in an out-of-order speculative CPU. ... In such a processor, any **instruction** within the **instruction window** can be executed when its operands are ready.

14.8

Register renaming is a form of pipelining that deals with data dependences between instructions by **renaming their register** operands. An assembly language programmer or a compiler specifies

these operands using architectural **registers** - the **registers** that are explicit in the instruction set architecture.

14.9:

(see link):

http://www.chegg.com/homework-help/key-elements-superscalar-processor-organization-chapter-16-problem-9rq-solution-9780134102061-exc

14.3 Consider the following assembly language program:

```
I1: Move R3, R7        /R3 ← (R7)/
I2: Load R8, (R3)      /R8 ← Memory (R3)/
I3: Add R3, R3, 4      /R3 ← (R3) + 4/
I4: Load R9, (R3)      /R9 ← Memory (R3)/
I5: BLE R8, R9, L3     /Branch if (R9) > (R8)/
```

This program includes WAW, RAW, and WAR dependencies. Show these.

Decode		Execute			Write		Cycle
I1	I2						1
	I2			I1			2
	I2			I1			3
I3	I4		I2				4
I5	I6		I4	I3	I1	I2	5
I5	I6	I5		I3			6
		I5	I6		I3	I4	7
							8
					I5	I6	9

Figure 14.13 An In-Order Issue, In-Order-Completion Execution Sequence

A:

True data dependency (RAW): I1 - I2, I1 - I3; I3 - I4, I2 - I5, I4 - I5

Output dependency (WAW): I1 - I3

Antidependency (WAR): I2 - I3

14.4 **a.** Identify the write-read, write-write, and read-write dependencies in the following instruction sequence:

```
I1:  R1 = 100
I2:  R1 = R2 + R4
I3:  R2 = r4 - 25
I4:  R4 = R1 + R3
I5:  R1 = R1 + 30
```

b. Rename the registers from part (a) to prevent dependency problems. Identify references to initial register values using the subscript "a" to the register reference.

A:

1. Identify the write-read, write-write, and read-write dependencies in the instruction sequence below by entering each line pair with a dependency in the correct column of the table to the right. For example, if L1 and L4 had a write-write dependency (which they don't), you would enter L1-L4 in the column labeled "write-write".

```
L1:  R1 = 100

L2:  R1 = R2 + R4

L3:  R2 = R4 - 25

L4:  R4 = R1 + R3

L5:  R1 = R1 + 30
```

write-read	write-write	read-write
*L2 – L4	*L1 – L2	L2 – L3
*L2 – L5	*L2 – L5	L2 – L4
L1 – L4	L1 – L5	L3 – L4
L1 – L5		L4 – L5

2. Rename the registers from problem 1 to prevent dependency problems. Identify references to initial register values using the subscript 'a' to the register reference.

```
L1:  R1b = 100

L2:  R1c = R2a + R4a

L3:  R2b = R4a - 25

L4:  R4b = R1c + R3a

L5:  R1d = R1c + 30
```

14.5 Consider the "in-order-issue/in-order-completion" execution sequence shown in Figure 14.13.

a. Identify the most likely reason why I2 could not enter the execute stage until the fourth cycle. Will "in-order issue/out-of-order completion" or "out-of-order issue/out-of-order completion" fix this? If so, which?

b. Identify the reason why I6 could not enter the write stage until the nineth cycle. Will "in-order issue/out-of-order completion" or "out-of-order issue/out-of-order completion" fix this? If so, which?

A:

a. From the drawing, it appears that I2 shares the same execute resources as I1. Therefore, I2 must wait until I1 is completed before it can be executed. Note that a resource conflict has the same result as a data dependency. The difference is that with a data dependency, the resource is data, not a physical device.

b.

• In in-order-issue, in-order-completion instruction “I5” has to completed before “I6” comes to the write stage and instructions “I5 and I6”

• In in-order-issue the instructions “I5 and I6” together should come out from the pipeline stages.

• In case of out-of-order issue it’s not the case where together can come out from the pipeline stages.

• So both the issue policies (in-order-issue out-of-order completion & out-of-order-issue out-of-order completion) will fix this.

Suppose if we consider out-of-order issue, out-of-order completion, then

• There exists true data dependency between instruction “I1 and I2”.

• Instruction “I2” must wait for “I1” to complete its process. So, “i2” stays in instruction window till “I1” comes out from the execution stage.

• Advantage of using “out-of-order issue, out-of-order completion” policy is as soon as resources are available; the instruction from instruction window comes to the execution stage.

• The instruction stays in the window till true data dependency between instructions is resolved.

From the block diagram given,

• Instruction “I2” waits in the window till “I1” is executed.

• “I2” exits from instruction window as “I1” frees up the resources “I2” needs.

• When “I2” completes execution process “I4” enters into execution stage.

• Instruction “I5” proceeds with the execution process without staying in instruction window.

• “I6” has to wait for “I4” to be completed.

Week 5 (Ch. 14 <Lab 4>)

14.6:
a and b both relate to pipelining dependencies and stuff (you can do this Travis, come on…)

Week 6 (Ch. 15 <Lab 5>)

Review Questions

15.1 Explain the distinction between the written sequence and the time sequence of an instruction.
15.2 What is the relationship between instructions and micro-operations?
15.3 What is the overall function of a processor's control unit?
15.4 Outline a three-step process that leads to a characterization of the control unit.
15.5 What basic tasks does a control unit perform?
15.6 Provide a typical list of the inputs and outputs of a control unit.
15.7 List three types of control signals.
15.8 Briefly explain what is meant by a hardwired implementation of a control unit.

A:

15.1

The operation of a computer, in executing a program, consists of a sequence of instruction cycles, with one machine instruction per cycle. This sequence of instruction cycles is not necessarily the same as the written sequence of instructions that make up the program, because of the existence of branching instructions. The actual execution of instructions follows a time sequence of instructions.

15.2

A micro-operation is an elementary CPU operation, performed during one clock pulse. An instruction consists of a sequence of micro-operations.

15.3

1. Define the basic elements of the processor. **2.** Describe the micro-operations that the processor performs. **3.** Determine the functions that the control unit must perform to cause the micro-operations to be performed.

15.4

Sequencing: The control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed. **Execution:** The control unit causes each micro-operation to be performed.

15.5

(1) Those that activate an ALU function. **(2)** those that activate a data path. **(3)** Those that are signals on the external system bus or other external interface.

15.6

The **inputs** are: **Clock:** This is how the control unit "keeps time." The control unit causes one micro-operation (or a set of simultaneous micro-operations) to be performed for each clock pulse. This is sometimes referred to as the processor cycle time, or the clock cycle time. **Instruction register:** The opcode of the current instruction is used to determine which micro-operations to perform during the execute cycle. **Flags:** These are needed by the control unit to determine the status of the processor and the outcome of previous ALU operations. **Control signals from control bus:** The control bus portion of the system bus provides signals to the control unit, such as interrupt signals and acknowledgments. The **outputs** are: **Control signals within the processor:** These are two types: those that cause data to be moved from one register to another,

and those that activate specific ALU functions. **Control signals to control bus:** These are also of two types: control signals to memory, and control signals to the I/O modules.

15.7

(1) Those that activate an ALU function. **(2)** those that activate a data path. **(3)** Those that are signals on the external system bus or other external interface.

15.8

In a hardwired implementation, the control unit is essentially a combinatorial circuit. Its input logic signals are transformed into a set of output logic signals, which are the control signals.

15.1 Your ALU can add its two input registers, and it can logically complement the bits of either input register, but it cannot subtract. Numbers are to be stored in two's complement representation. List the micro-operations your control unit must perform to cause a subtraction.

A:

Consider the instruction SUB R1,X which subtract the contents od location X from the contents of register R1, and place the result in R1.

T1: MBR<- IR(address)
T2: MBR<-Memory
T3: MBR<-complement (MBR)
T4: MBR<-Increment (MBR)
T5: R1<-R1 + (MBR)

15.2 Show the micro-operations and control signals in the same fashion as Table 15.1 for the processor in Figure 15.5 for the following instructions:

- Load Accumulator
- Store Accumulator
- Add to Accumulator
- AND to Accumulator
- Jump
- Jump if AC = 0
- Complement Accumulator

A:

(This is easy Travis, you can figure this out…)

15.4 Write the sequence of micro-operations required for the bus structure of Figure 15.6 to add a number to the AC when the number is

a. an immediate operand
b. a direct-address operand
c. an indirect-address operand

A:

i. an immediate operand

t1 : Y ← (IR(address))
t2 : Z ← (AC) + (Y)
t3 : AC ← (Z)

ii. a direct-address operand

t1: MAR ← (IR(address))
t2: MBR← Memory
t3: Y ← (MBR)
t4: Z ← (AC) + (Y)
t5: AC ← (Z)

iii. an indirect-address operand

t1: MAR← (IR(address))
t2: MBR←Memory
t3: MAR← (MBR)
t4: MBR←Memory
t5: Y← (MBR)
t6: Z← (AC) + (Y)
t7: AC← (Z)

Week 7 (Ch. 16 <Lab 6>)

Review Questions

16.1 What is the difference between a hardwired implementation and a microprogrammed implementation of a control unit?
16.2 How is a horizontal microinstruction interpreted?
16.3 What is the purpose of a control memory?
16.4 What is a typical sequence in the execution of a horizontal microinstruction?
16.5 What is the difference between horizontal and vertical microinstructions?
16.6 What are the basic tasks performed by a microprogrammed control unit?
16.7 What is the difference between packed and unpacked microinstructions?
16.8 What is the difference between hard and soft microprogramming?
16.9 What is the difference between functional and resource encoding?
16.10 List some common applications of microprogramming.

A:

16.1

A hardwired control unit has a processor that generates signals or instructions to be implemented in correct sequence. This was the older method of control that works through the use of distinct components, drums, a sequential circuit design, or flip chips.

A micro programmed control unit on the other hand makes use of a micro sequencer from which instruction bits are decoded to be implemented. It acts as the device supervisor that controls the rest of the subsystems including arithmetic and logic units, registers, instruction registers, off-chip input/output, and buses.

16.2

1. To execute a microinstruction, turn on all the control lines indicated by a 1 bit; leave off all control lines indicated by a 0 bit. The resulting control signals will cause one or more micro-operations to be performed. **2.** If the condition indicated by the condition bits is false, execute the next microinstruction in sequence. **3.** If the condition indicated by the condition bits is true, the next microinstruction to be executed is indicated in the address field.

16.3

The control memory contains the set of microinstructions that define the functionality of the control unit.

16.4

The microinstructions in each routine are to be executed sequentially. Each routine ends with a branch or jump instruction indicating where to go next.

16.5

In a **horizontal microinstruction** every bit in the control field attaches to a control line. In a **vertical microinstruction**, a code is used for each action to be

16.6

Microinstruction sequencing: Get the next microinstruction from the control memory.
Microinstruction execution: Generate the control signals needed to execute the microinstruction.

16.7

The degree of packing relates to the degree of identification between a given control task and specific microinstruction bits. As the bits become more **packed**, a given number of bits contains more information. An unpacked microinstruction has no coding beyond assignment of individual functions to individual bits.

16.8

Hard microprograms are generally fixed and committed to read-only memory. **Soft microprograms** are more changeable and are suggestive of user microprogramming.

16.9

Two approaches can be taken to organizing the encoded microinstruction into fields: functional and resource. The **functional encoding** method identifies functions within the machine and designates fields by function type. For example, if various sources can be used for transferring data to the accumulator, one field can be designated for this purpose, with each code specifying a different source. **Resource encoding** views the machine as consisting of a set of independent resources and devotes one field to each (e.g., I/O, memory, ALU).

16.10

Realization of computers. Emulation. Operating system support. Realization of special-purpose devices. High-level language support. Microdiagnostics. User Tailoring.

Other Stuff

Average memory access time:
http://www.cs.fsu.edu/~hawkes/cda3101lects/chap7/avgmemaccesstime.html