CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce

343 Davis Hall

alphonce@buffalo.edu

Office hours:

Tuesday 10:00 AM – 12:00 PM*

Wednesday 4:00 PM – 5:00 PM

Friday 11:00 AM – 12:00 PM

OR request appointment via e-mail

**Tuesday adjustments: 11:00 AM – 1:00 PM on 10/11, 11/1 and 12/6*

ANNOUNCEMENTS

ANNOUNCEMENTS

Recitations start this week (in Baldy 21)

Bring your UB card

Main course website:

www.cse.buffalo.edu/faculty/alphonce/cse115/

READINGS

Quick overview on the weekend.

Revisit in detail throughout the week.

Do embedded exercises to check your understanding. No set due-date, but keep up so you don't fall behind.

Moving forward, I will generally post readings for the upcoming week by Thursday evening.

ROADMAP

Last time

- Low-level issues

Today

- Expressions and objects
- Memory diagrams

Coming up

- Class definitions
- Variables
- Method calls
- Object diagrams

PROFESSIONALISM

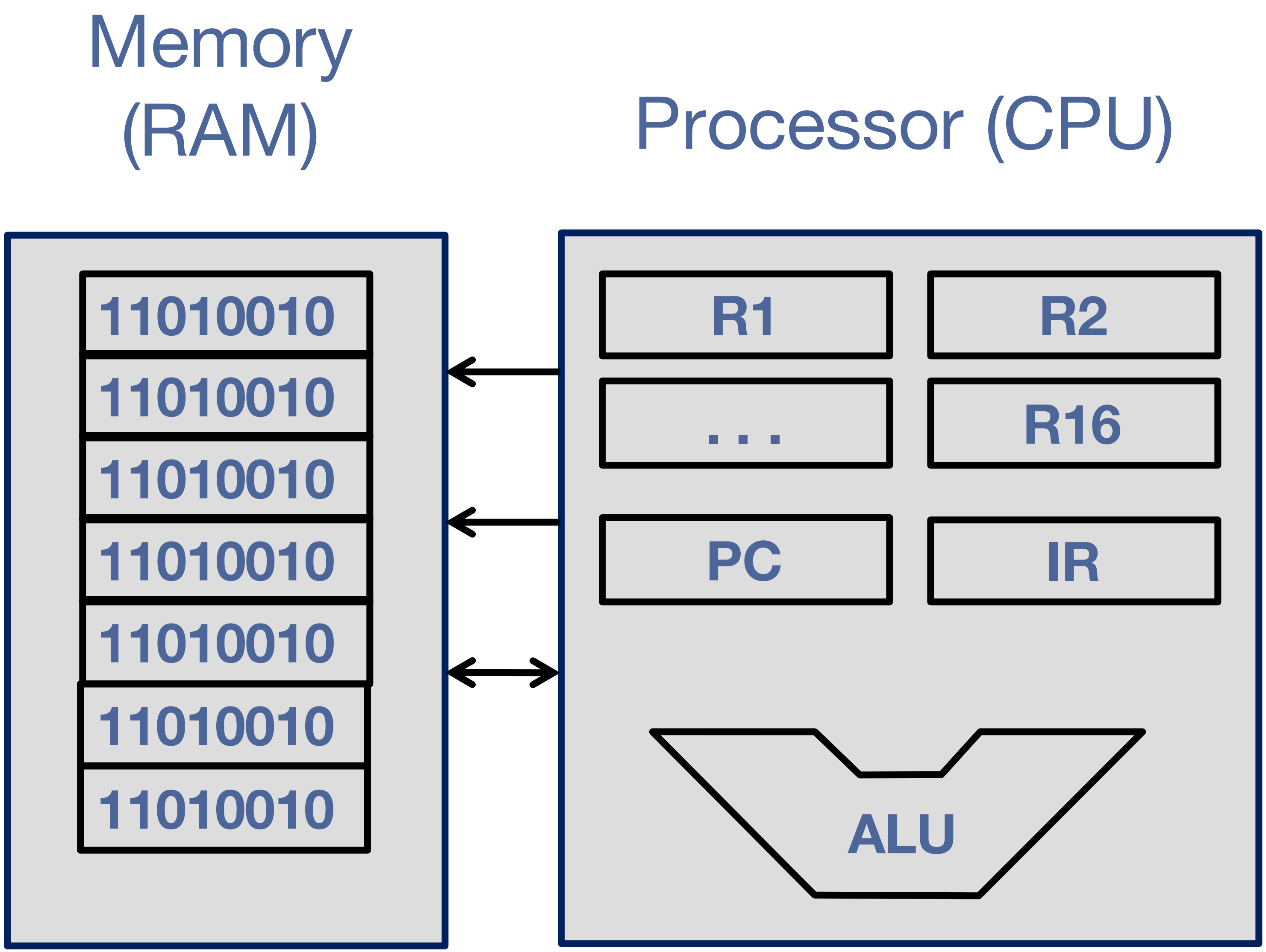
Please turn off and put away electronics:

- cell phones
- pagers
- laptops
- tablets
- etc.

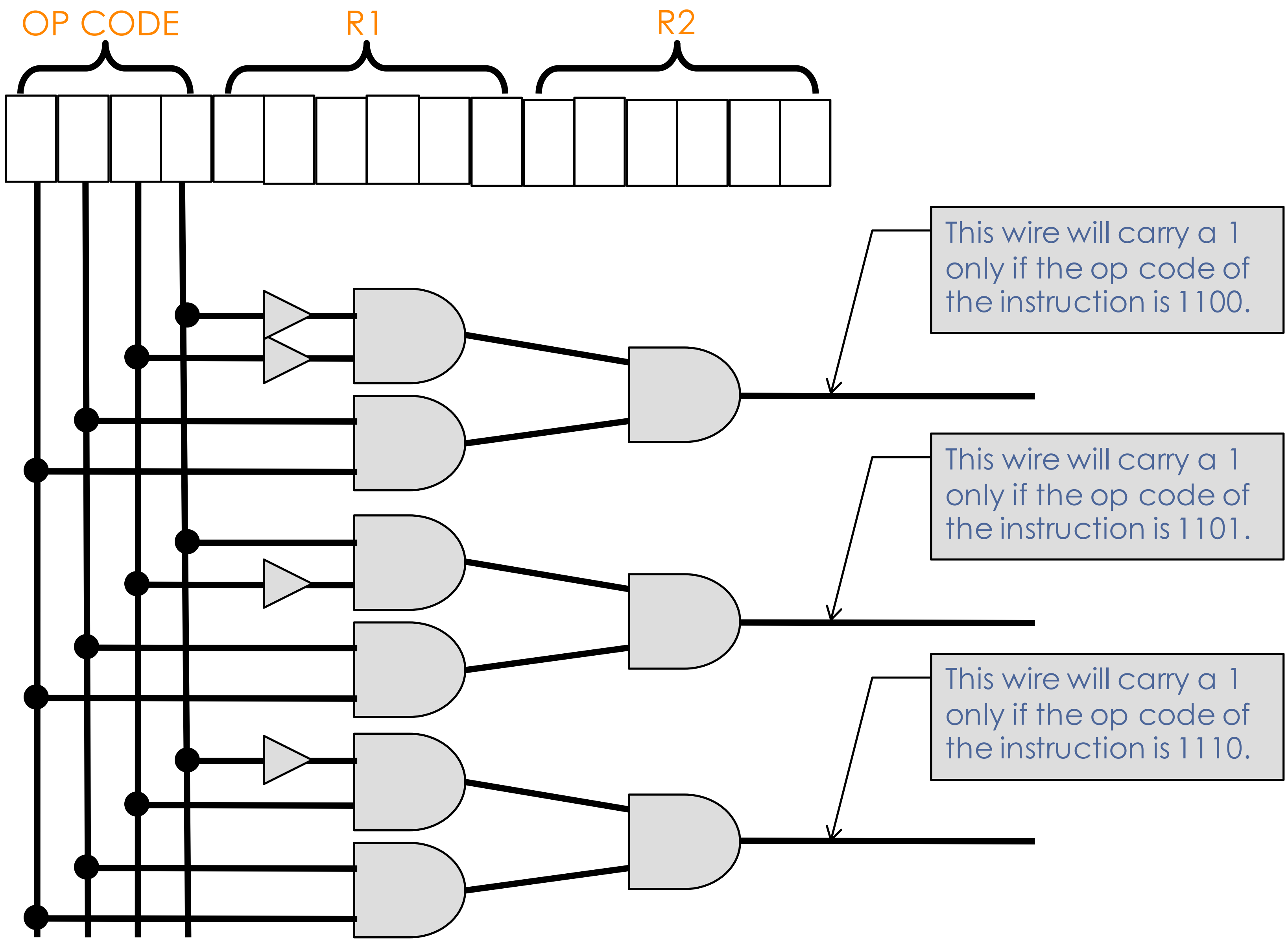
REVIEW

INSTRUCTION DECODING

Computer Organization



Instruction decoding



FETCH
DECODE
EXECUTE

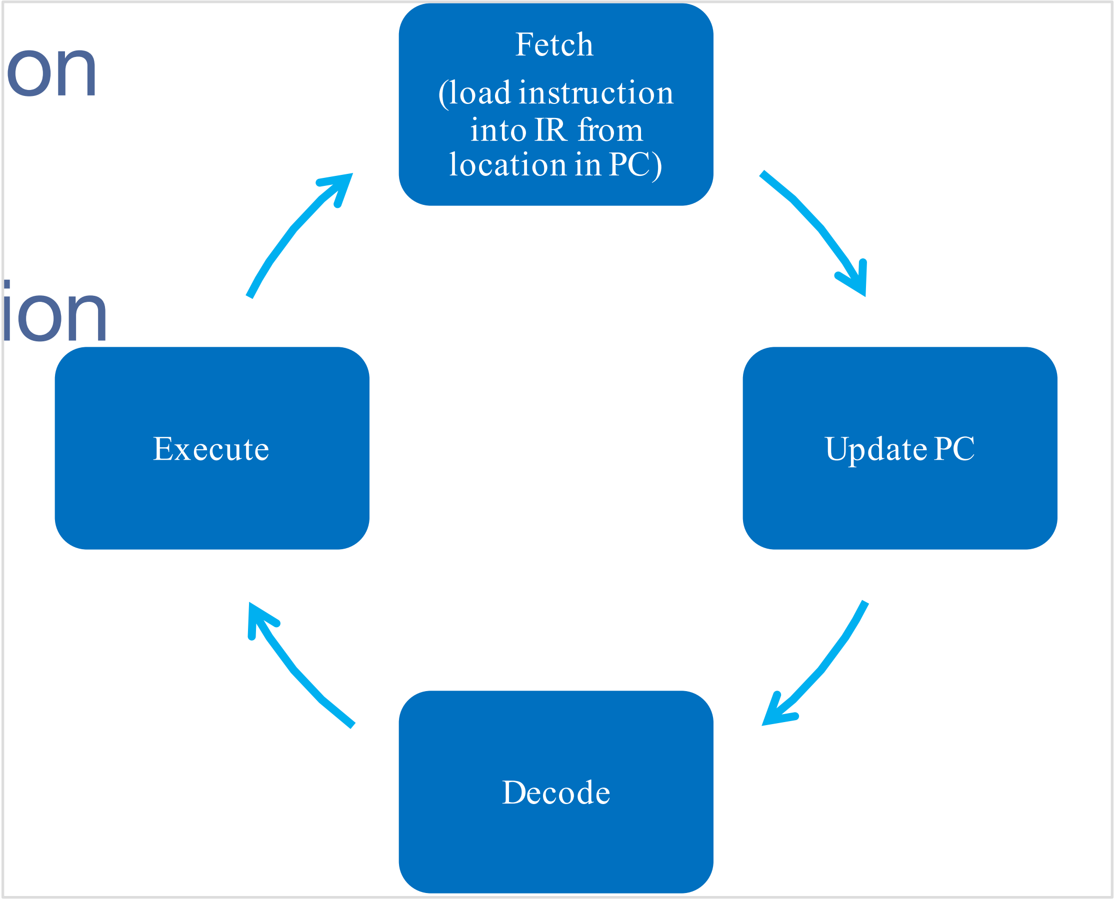
cycle

Fetch-Decode-Execute cycle

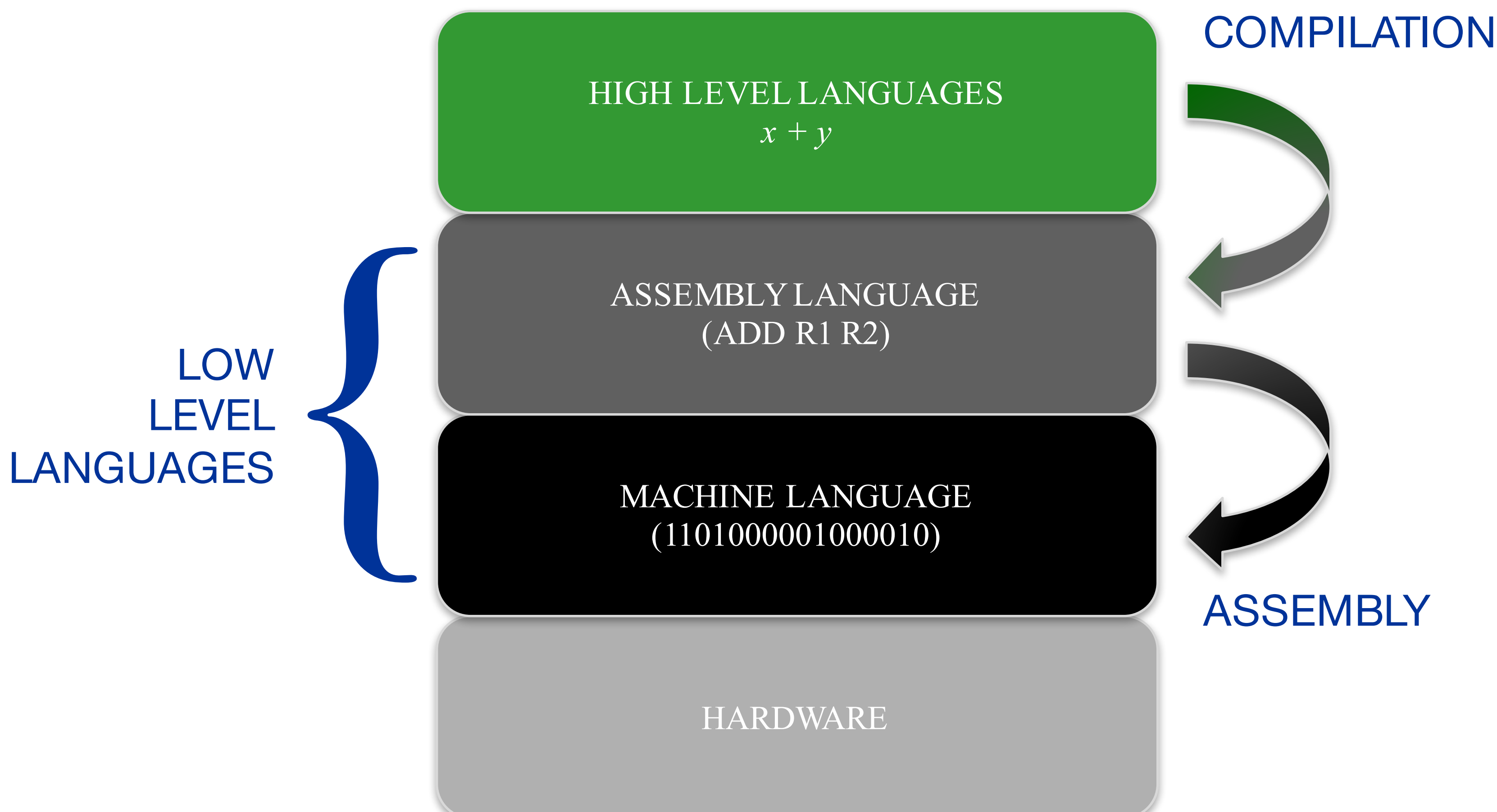
Fetch an instruction (& update PC)

Decode instruction

Execute instruction



Language levels



MOVING ON

Question

Is every formal language a “programming language”?

In other words, can any formal language be used to solve any computational problem?

No.

What makes a language a programming language? (Böhm-Jacopini theorem, 1966)

Sequencing

Selection

Repetition

Sequencing

the language must permit the order of instructions to be specified

Selection

the language must permit different instructions to be executed based on the outcome of a decision

Repetition

the language must permit an instruction to be executed repeatedly, based on the the outcome of a decision

Equivalences

Computation models

- Turing Machine (en.wikipedia.org/wiki/Turing_machine)
- Lambda calculus (en.wikipedia.org/wiki/Lambda_calculus)
- and others (en.wikipedia.org/wiki/Computable_function)

Examples of high-level programming languages

- Java
- C#
- Erlang
- Fortran
- Prolog
- Python
- Lisp
- ML
- Ruby

High-level languages

Richer syntax than

- Machine language (bit strings)
- Assembly language (mnemonic)

Improved readability/writeability

Must be translated (compiled) to machine language

Java

A modern high-level language

A (relatively) small and simple core language

Object-oriented

Large libraries

Moving on…:

We will return to low-level issues later in the semester, and also in later courses.

This brief low-level discussion gives context for upcoming topics.

Now we turn to some higher-level issues.

I have a question for you!

What did you have for breakfast today?

I have a question for you!

What did you have for breakfast today?

This exercise is due to Dr. Joe Bergin.

Activity

The goal of this short activity is to demonstrate two things:

1. objects have state
2. objects have identity
3. objects have behaviors
4. sending a message to an object can trigger one of its behaviors

Questions?

Objects

OO software systems are systems of interacting objects.

Objects have

properties:

these are things that objects know

e.g. what you had for breakfast

behaviors:

these are things objects do

e.g. being able to reply to the question “What did you have for breakfast?”

How do we create an object?

```
new example1.BarnYard()
```

There are three parts to this expression:

- new
- example1.BarnYard
- ()

Expression evaluation

evaluating `new example1.BarnYard()`

produces a value

as a *side effect* causes an object to be created and initialized

(part of) memory

107	
108	
109	
110	
111	
112	
113	
114	
115	

Understanding the *side effect*

At any given point in time some locations in memory are being actively used to hold information, while others are available for use.

For the sake of this example, let us assume that the memory locations with addresses 107 and 115 are in use, and locations with addressed 108 through 114 are available.

(part of) memory

107	used
108	available
109	available
110	available
111	available
112	available
113	available
114	available
115	used

evaluating a ‘new’ expression

When evaluating an expression like ‘new example1.BarnYard()’, the operator ‘new’ first determines the size of the object to be created (let us say it is four bytes for the sake of this example).

107	**used**
108	**available**
109	**available**
110	**available**
111	**available**
112	**available**
113	**available**
114	**available**
115	**used**

evaluating a ‘new’ expression

When evaluating an expression like ‘new example1.BarnYard()’, the operator ‘new’ first determines the size of the object to be created (let us say it is four bytes for the sake of this example).

Next, new must secure a contiguous block of memory four bytes large, to store the representation of the object.

107	used
108	reserved by ‘new’
109	reserved by ‘new’
110	reserved by ‘new’
111	reserved by ‘new’
112	available
113	available
114	available
115	used

evaluating a ‘new’ expression

When evaluating an expression like ‘new example1.BarnYard()’, the operator ‘new’ first determines the size of the object to be created (let us say it is four bytes for the sake of this example).

Next, new must secure a contiguous block of memory four bytes large, to store the representation of the object.

Bit strings representing the object are written into the reserved memory locations. In this example we use “10101010” to indicate that some bit string was written into a given memory location; the exact bit string written depends on the specific details of the object.

107	used
108	**10101010**
109	**10101010**
110	**10101010**
111	**10101010**
112	available
113	available
114	available
115	used

evaluating a 'new' expression

When evaluating an expression like 'new example1.BarnYard()', the operator 'new' first determines the size of the object to be created (let us say it is four bytes for the sake of this example).

Next, new must secure a contiguous block of memory four bytes large, to store the representation of the object.

Bit strings representing the object are written into the reserved memory locations. In this example we use "10101010" to indicate that some bit string was written into a given memory location; the exact bit string written depends on the specific details of the object.

The **starting address** of the block of memory holding the object's representation is the value of the 'new' expression. This address is called a '***reference***'.

107	used
108	10101010
109	10101010
110	10101010
111	10101010
112	available
113	available
114	available
115	used

Expression evaluation

evaluating `new example1.BarnYard()`

produces *a value* (which we call a reference)

causes a *side effect* (an object is created and initialized)

we can remember a reference value by storing it in a variable

The variable declaration

Variables must be <u>declared</u> before use

- declaration specifies encoding scheme
- declaration specifies size

Declaration consists minimally of

- <u>type</u>
- <u>name</u>

The semicolon ';' is a *terminator*.

Examples

```
example1.BarnYard    by ;
example1.Chicken      c ;
```

assignment statement

To associate a value with a variable, use an assignment statement:

SYNTAX: <variable> = <expression> ;

'=' is the ASSIGNMENT OPERATOR (it is not 'equals'!)

Example

```
by = new example1.BarnYard();
```

"by is assigned the value of the expression 'new example1.BarnYard()' " …or…

"by is assigned a reference to a new example1.BarnYard() object" …or…

"by is assigned a reference to a new BarnYard object" (example1 is implied)