

Welcome back²²
to CS429H!

Week 4



Ed memes of the week:



My P4 Link Register

Public Playlist

Prog Grind Playlist

Jay_Shin • 8 songs, 26 min 26 sec

Custom order

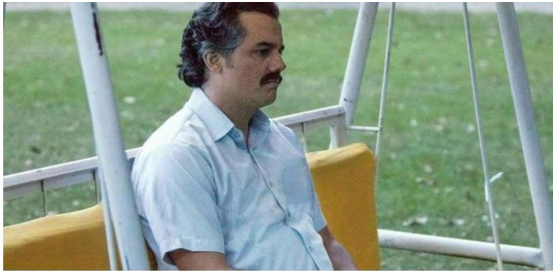
#	Title	Album	Date added	
1	I'm Done	I'm Done	57 seconds ago	2:29
2	my	my	45 seconds ago	3:17
3	Programs	Programs	39 seconds ago	3:30
4	No More	SANTIAGO	29 seconds ago	3:11
5	Falling Test Cases	Electronic	23 seconds ago	4:14
6	damd	damd	12 seconds ago	1:54
7	Failure	Dark Before Dawn	6 seconds ago	3:38
8	Segmentation Fault	Data Renaissance	0 seconds ago	4:50

C6.2.240 ORR (immediate)

```
= DecodeBitMasks(N, imms, immr, TRUE, datasize);
```



Gheith when he can't see us for two weeks.



other

- An instructor (Ahmed Gheith) endorsed this note -

vote for ur fav piazza shitpost:

A. covid spring break

Gheith's music preferences?

What music does Gheith like?

other

~ An instructor (Adam Schoenberg) endorsed this question ~

Edit undo good question | 22

S the students' answer, where students collectively construct a single answer

Click to start off the wiki answer

i the instructors' answer, where instructors collectively construct a single answer

Not(Ariana Grande) & Not(Lil Nas X)

~ An instructor (Nikita Sharma) endorsed this answer ~

vote for ur fav piazza shitpost:

- A. covid spring break
- B. gheith hates ariana grande

vote for ur fav piazza shitpost:

- A. covid spring break
- B. gheith hates ariana grande
- C. classic grade release timeline

To the TA's: is this a reasonable timeframe for grade release?

I propose the following schedule for grade release; TA's, please let us know if this sounds reasonable. We'd really like to at least get some sort of feedback before the next midterm. And as always, thanks for all of y'all's hard work in running discussions/office hours/making the course a really interesting one! :)

Proposed grade release times (feel free to modify as you see fit):

HW 2: April 3

HW 3: April 6

HW 4: April 10

HW 5: April 13

HW 6: April 17

HW 7: April 20

etc.

Of course, please take your time; this is just a rough estimate for what we think would be a nice timeframe. I know grading is definitely time-consuming, but y'all got this!

Where I'm at mentally



vote for ur fav piazza shitpost:

- A. covid spring break
- B. gheith hates ariana grande
- C. classic grade release timeline
- D. surfing the web

a modest proposal

so i've noticed some spirited discussion on being at the computer at the right time and trying to receive the lowest alias number possible by doing a small commit (eg touch spamon) and then pushing it the second it releases.

however, i have a modest proposal as a means of making this a bigger challenge: release the project at 4:30 AM

this way, the truly determined individual gets the number they deserve

thank you for coming to my ted talk

vote for ur fav piazza shitpost:

- A. covid spring break
 - B. gheith hates ariana grande
 - C. classic grade release timeline
 - D. surfing the web
 - E. screaming thread
 - F. a modest proposal
-

Stress

- 429H is not an easy class
 - Lots of new materials
 - Unfamiliar programming environments
 - Fast, often relentless pace
- Struggling in this course is normal
 - There will be times you won't know the answer of the solution
 - This is expected—we want everyone to succeed, but the only way we can help is if you ask for it
- If you find yourself overly overwhelmed or spending more time on this class than you think you should be, please reach out to Dr. Gheith or the TAs
 - We can help out as far as the class goes
 - We can provide other resources where we are not able to help

[Mental health resource available at UT](#)

Questions on lecture content?
Or about cats?

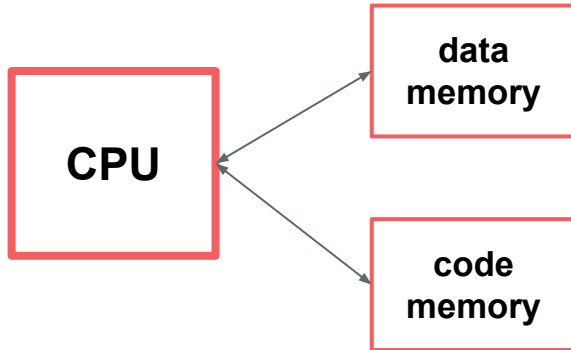
A Note on Regrades (Reminder)

- **PLEASE DO NOT** tell us to grade a different commit before the grades are released
- We will default to grading the last commit before the **soft** deadline
- We will **NOT** grade a late submission unless you ask us to (see below)
- We will **NOT** grade a late commit for a test case
- Regrade requests:
 - If grades are released and your last commit before the soft deadline broke something that was working in a previous commit
 - You got an extension
 - You want us to grade a late submission (**50% penalty on entire project grade in this case**)

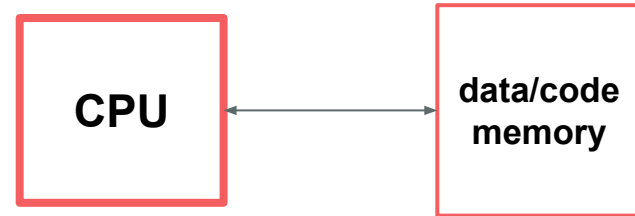
Quiz ^{review} everyone say AWWW!

Question 1

Harvard Architecture:

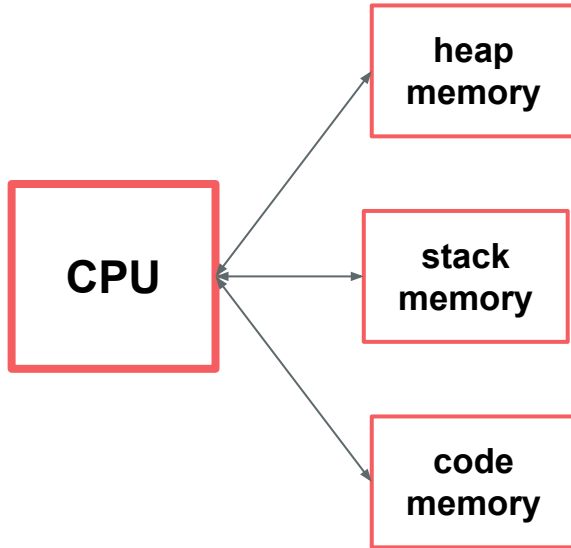


Von Neumann Architecture:



Question 1

Hydra™ Architecture: *(fictional)*



- What is a **tradeoff**?
- What are some tradeoffs of Hydra vs. Von Neumann vs. Harvard?

Question 2

[2 Points] Evil Gheith has removed the ADD instruction. Name two strategies to replace it? Give a tradeoff between the two different strategies.

- Subtract the 2s Complement
- Perform the boolean logic on the registers
- Loop Increment

Question 3

3. **[2 points]** Recall the three addressing modes we have in AArch64 for load and store instructions. Are all these addressing modes necessary? Suppose the architecture only supported unsigned offset.
 - a) Name one advantage and one disadvantage of this.

Question 3

Advantages of only using unsigned offset

- Would simplify decoding (because less possible options) which would lead to less transistors, increasing energy efficiency and space availability.
- More readable, clearer when registers are actually being changed

Disadvantages:

- Need 2 instructions instead of 1, which wastes space in memory and disk, and could also be less efficient
- Putting it into one instruction guarantees it will not be interrupted in the middle. Having an interrupt in between the two instructions may lead to some unintended consequences, such as having the stack in an inconsistent state

Question 3

- b) Several fairly common programming paradigms would motivate the desire for pre and post-indexing addressing modes. Give an example of some code that could be optimized by using each of these addressing modes.

Pre-indexing:

`a[++i]`

Post-indexing:

`a[i++]`

Question 4

[2 points] ARM and x86 take different approaches to how functions are called. In ARM, the BL instruction will store the return address in the Link Register, and the RET instruction is like a jump to an address in a register, with extra semantic info that it is “returning”. In x86, the CALL instruction pushes the return address to the stack, and the RET instruction pops from the stack into the program counter. What is an advantage of each system?

Question 4

[2 points] ARM and x86 take different approaches to how functions are called. In ARM, the BL instruction will store the return address in the Link Register, and the RET instruction is like a jump to an address in a register, with extra semantic info that it is “returning”. In x86, the CALL instruction pushes the return address to the stack, and the RET instruction pops from the stack into the program counter. What is an advantage of each system?

ARM: optimized for leaf function calls, RISC

x86: optimized for nested function calls, simpler assembly/machine code (fewer instructions)

Common mistakes - LR only used for return address (can be GPR x30), have to pop from stack to get value (can do random access), x86 has an extra register (LR is GPR, also x86 has 16 GPRs while ARM has 32), impossible to clobber (less predictable where return address will end up if it is pushed to stack, but can still overwrite it once it's there)

Question 5

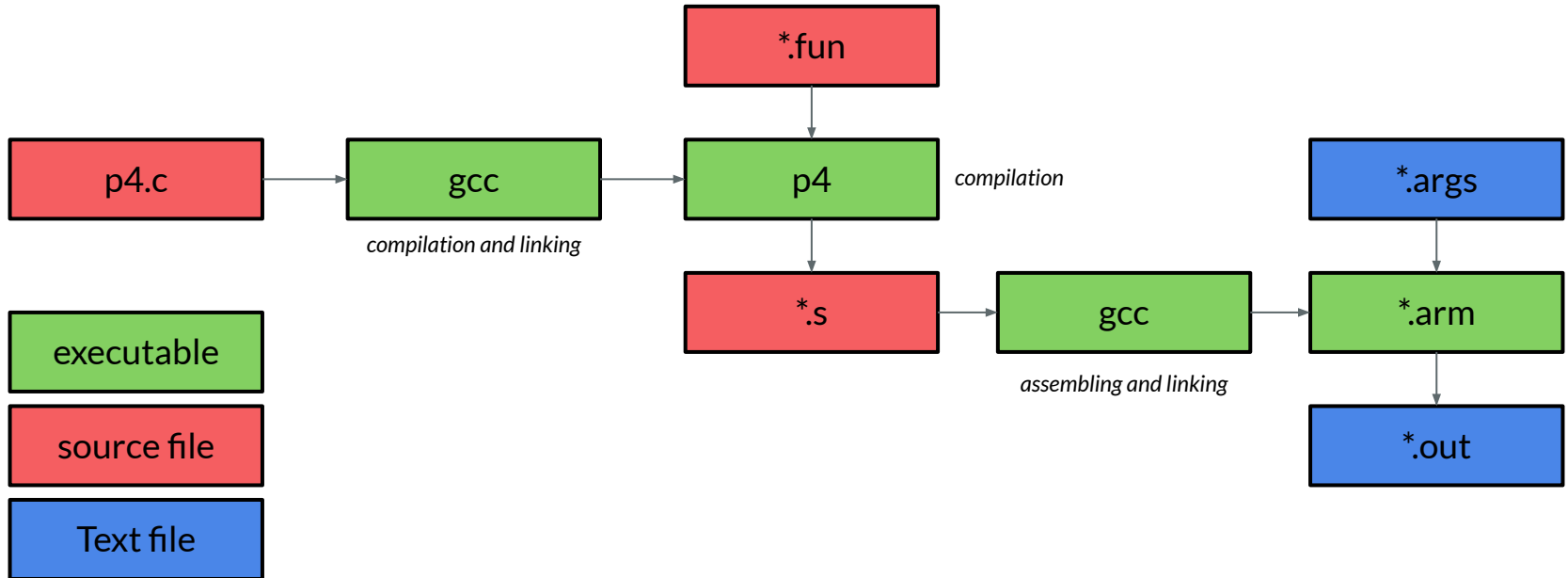
Expression	Value
<code>sizeof(Person)</code>	0x8
<code>sizeof(stuff)</code>	0x8
<code>&(stuff->firstInitial)</code>	0x1234
<code>&(stuff->lastInitial)</code>	0x1235
<code>stuff + 0x1</code>	0x123C
<code>&(stuff[1].id)</code>	0x1240
<code>&(1[stuff].id)</code>	0x1240
<code>&(stuff[-2])</code>	0x122A
<code>&(stuff->id) - &(stuff->age)</code>	0x2 (this part is bad and everyone gets credit)

P4

(for real this time)

(and we actually converted everything to ARM)

what is p4?



Compilers are complicated

How do you map variables to 32 registers & memory locations?

For example, `int c = a + b;`

- Where is a? b? Where should c go? Are both a and b only in memory? What regs can we modify?

p4 pro tip – do not hold values in registers

How do you map variables to ~~32 registers~~ & memory locations?

For example, `int c = a + b;`

- Where is a? b? Where should c go? Are both a and b only in memory? What regs can we modify?

p4 pro tip — use comments in your generated Assembly!

Just like comments in C!

Can use

```
//
```

and

```
/* ... */
```

Stack Machines

Let's say our architecture has only one general purpose register: %rsp. To make up for this, we are changing the ISA to only include the following instructions:

```
PUSH val // pushes a value onto the stack
```

```
ADD // pops 2 values, adds them, and pushes the result
```

```
NEGATE // pops a value, negates it, and pushes the result
```

```
PRINT // pops from the stack and prints
```

Calling Convention

- C ABI for functions defines which registers are for parameters and returning
- Only necessary to call external functions (that you don't compile)
- Calling your own functions can use whatever convention you want
 - Can you change your convention based on anything?
 - Does it have to be consistent with itself?
 - What are some tradeoffs of staying true to the ARM calling convention?

Using Labels

```
.section .data
variable: .quad 0x0123456789ABCDEF
foo:      .quad 0x0
```

```
.section .text
fun1: ...
```

```
main:
```

```
    adrp    x0, fun1
    add     x0, x0, :lo12:fun1
    adrp    x1, foo
    add     x1, x1, :lo12:foo
    str     x0, [x1]
    ldr     x1, [x1]
    mov     x0, #0
    blr     x1
    adrp    x1, variable
    add     x1, x1, :lo12:variable
    str     x0, [x1]
```

```
foo = fun {...}
variable = foo(0)
```

Printing Things!

What does this do?

```
//data segment  
puts("    .data");  
puts("format: .byte '%', '\l', 'u', 10, 0");
```

Printing Things!

```
$ cat test.c
```

```
#include <stdio.h>
```

```
int main(int argc, char **argv) {
```

```
    printf("%lu\n", argc);
```

```
    return 0;
```

```
}
```

```
$ gcc test.c
```

```
//data segment  
puts("    .data");  
puts("format: .byte '%', '\l', 'u', 10, 0");
```

```
$ objdump -d a.out
```

```
...
```

```
0000000004005d4 <main>:
```

```
4005d4: a9be7bfd stp x29, x30, [sp, #-32]!
```

```
4005d8: 910003fd mov x29, sp
```

```
4005dc: b9001fe0 str w0, [sp, #28]
```

```
4005e0: f9000be1 str x1, [sp, #16]
```

```
4005e4: b9401fe1 ldr w1, [sp, #28]
```

```
4005e8: 90000000 adrp x0, 400000 <__abi_tag-0x254>
```

```
4005ec: 911ac000 add x0, x0, #0x6b0
```

```
4005f0: 97ffffa8 bl 400490 <printf@plt>
```

```
4005f4: 52800000 mov w0, #0x0 // #0
```

```
4005f8: a8c27bfd ldp x29, x30, [sp], #32
```

```
4005fc: d65f03c0 ret
```

```
...
```

```
$ readelf -p .rodata a.out
```

```
String dump of section '.rodata':
```

```
[ 18] %lu^J
```


How to debug compiled test cases?

```
~gheith/public/qemu_5.1.0_old/bin/qemu-aarch64 -g 1234  
./file.arm > ./file.out
```

In another terminal window:*

```
~gheith/public/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linu  
x-gnu/bin/aarch64-none-linux-gnu-gdb ./file.arm
```

```
> target remote localhost:1234
```

```
> b main
```

```
> continue
```

*As of Thursday, Gheith's ARM GDB was not working after a dependency was updated on the lab machines. He is looking into it

Change your Makefile for ARM Debugging Symbols

Change Line 67 of your Makefile. Add a -g file so that Line 67 reads

```
-${ARM GCC} -static -g -o $*.arm $*.s
```

How to debug compiled test cases (Elie's Version)

```
~gheith/public/qemu_5.1.0_old/bin/qemu-aarch64 -g 1234  
./file.arm > ./file.out
```

In another terminal window:

```
~elies/public/bin/gdb <file.arm>
```

```
> target remote localhost:1234
```

```
> b main
```

```
> continue
```

debugging a compiler?

how to use gdb with assembly?

- `layout asm` → like ``layout src`` but for assembly
- `ni` → like ``next`` but instead of next statement, it goes to the next instruction
- `si` → like ``step`` but instead of stepping into statements, it steps into calls and jumps
- `info reg` → display the contents of all the registers
- `tui reg general`

P5

How to Heap

- Heap is hard
- *Consistency*
- Keep consistency through *invariants*
 - An *invariant* should be true at the **beginning** and **end** of all heap functions
 - They can be violated *temporarily* in the middle of these functions
 - Examples of invariants?

How to Heap


- Structs and Functions! Strunctions!
- What kind of structs might you need?
- What kind of functions might you need?
- What kind of strunctions might you need?

How to Heap

```
struct __attribute__((packed)) foo {  
    int a : 2;  
    int b : 6;  
};
```

- What is sizeof(struct foo)?
- Why might you want to do this?

Debugging Tips

- How can you check your invariants?
- **Diagnostics**—use a `heap_check()` function
 - Pretty-print entire heap state
 - Check invariants programmatically
 - Call after every `malloc/free`
 - + Catch bugs early
 - - Makes your code slooow 
 - Call only in certain cases
 - e.g. (gdb) call `print_heap()`
 - + Less verbose / spammy
 - - Lower coverage
- **Downsize the test case**
 - Small test cases are easier to debug
- **Debug interactively with gdb!**
 - `watch` and `rwatch`
 - (gdb) `watch head`
 - (gdb) `rwatch (long *) 0x832a8b0`
 - (gdb) `watch curr_block->free`

Debugging Tips

```
CFLAGS = -Werror -Wall -O3 -g -std=c11
```

changes to

```
CFLAGS = -Werror -Wall -O0 -g -std=c11
```

Why can't we use printf debugging for the heap?

Debugging Tips - Conditional Compilation

```
CFLAGS = -Werror -Wall -O3 -g -std=c11
```

changes to

```
CFLAGS = -Werror -Wall -O3 -g -std=c11 -DDEBUG
```

Then in your code, you can have

```
#ifdef DEBUG
```

```
heap_check()
```

```
#endif
```

Questions?

