Recitation 11/13

hello to anyone watching the recording :))

Important Note:

If you want to reference the single cycle processor diagram, do NOT use past slides (for both lec and recitation) - the diagram has been since updated to correct a few errors

All your reference sheets should be from the course website - if there are any conflicts between those and the slides, trust the ref sheets over the slides

Practice (Conceptual) Questions

1. What do we know about the time duration of one cycle in our risc-v single cycle processor?
2. What is the difference between PCAddMux.CTL and PCMux.CTL?
3. What are U type instructions? Why are they necessary?
    a. Hint: what is the largest immediate we can use in an addi instruction?
4. Why do we save PC+4 in a register when we call jalr/jal?
    a. Hint: why is it called “link” in “jump and link?” What is being linked?
5. If we were to assign a type (eg. int, char, char*) to PC, what type would it be?
6. Why is it important to perform sign extensions on the immediates before we perform the intended operation in the ALU? (i.e. sign extend immediate before adding it to rs1 in the addi instruction)

Endianness

If reading from left to right:

- Little endian: least significant byte is read first
- Big endian: most significant byte read first

Our eyes would have an easier time interpreting big endian byte ordering

- But our local machines use little endianness
- Network byte order is big endian

Endianness Example

int x = 0x12345678

In big endian: 12 34 56 78

address	0	1	2	3
Value (hex)	12	34	56	78

In little endian: 78 56 34 12

address	0	1	2	3
Value (hex)	78	56	34	12

Endianness - why are there two?

advantages and disadvantages when reading memory from left to right

Big Endianness:

- First byte indicates if value is positive or negative
- Easier to determine magnitude of the value

Little Endianness:

- Bit arithmetic is performed from LSB moving the carry-over towards the MSB, more efficient on little-endian systems
- Can interpret small values i.e. 0x4A000000 as the same value reading only 1 byte (0x4A), 2 bytes (0x004A), or 4 bytes without recalculating address
    - Potential avenue for code optimization by low-level (asm) programmers

This shows: there are lots of deliberate choices to be made about how information (stored in 1’s and 0’s) can be interpreted!

Any questions on the following functions?

C Stream Functions (1 of 3)

- Some stream functions (complete list in `stdio.h`):

Returns NULL on error

Do we create a new file if it doesn't exist?
Are we reading the file?
Are we writing the file?

```
FILE* fopen(filename, mode);
```

- Opens a stream to the specified file in specified file access mode

a **FILE*** returned by fopen

```
int fclose(stream);
```

- Closes the specified stream (and file)

```
int fprintf(stream, format, ...);
```

- Writes a formatted C string like `printf(...);` but for files

```
int fscanf(stream, format, ...);
```

- Reads data and stores data matching the format string

C Stream Functions (2 of 3)

- Some stream functions (complete list in `stdio.h`):

Pointer to the start of elements in memory to write to file

Size of an element

Number of elements

`FILE*`

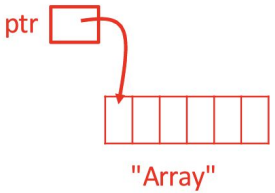
```
size_t fwrite(ptr, size, count, stream);
```

- Writes an array of *count* elements of *size* bytes from *ptr* to *stream*

```
size_t fread(ptr, size, count, stream);
```

- Reads an array of *count* elements of *size* bytes from *stream* to *ptr*

Returns number of elements actually read/written



C Stream Functions (3 of 3)

- Some stream functions (complete list in `stdio.h`):

```
int fgetc(FILE *stream);
```

- Reads one character from stream (one byte)

```
int fputc(FILE *stream);
```

- Writes one character from stream (one byte)

```
char* fgets(char* str, int n, FILE* stream);
```

- Reads a string from the stream into the string **`str`**. Reads N characters or until a newline character (or end of file).

C Stream Error Checking/Handling

- Some error functions (complete list in `stdio.h`):

```
int ferror(FILE *stream);
```

  - Checks if the error indicator associated with the specified stream is set

```
int clearerr(FILE *stream);
```

  - Resets error and EOF indicators for the specified stream

```
int perror(char *s);
```

  - Prints `message` followed by an error message related to `errno` to `stderr`

Global variable

Extra information

Other Functions

- Many other functions not covered in lecture (not enough time). Feel free to look up others and use them

- Some examples:
  - `int feof(FILE* f);`
    - check for end of file
  - `void rewind(FILE *f);`
    - start back at the beginning of file
  - `long ftell(FILE* f);`
    - gives the current position into the file
  - `int fseek(FILE* f, long offset, int whence);`
    - Reposition where we are in the file