Group #		7
Grader:	jangarit	
Total Grade:		173
Part 1		83
Part 2		90
Late Penalty		0

Point Break Down Key	
B	Binary - Take whole amount or Give whole amount
S	Split - take or give a portion of total OR give/take total

Lab 1 Grading Rubric

PART 1				Notes
2.6 Revision Control System (15 pts)	**Possible Points**	**Point Break Down**	**Points Awarded**	
Discounting hidden files, what are the files that are in the repository? What are their names? What are their contents?	8	S	8	- 'crimpers' has "42" (2 + 2 points) - 'wire' has 'strippers' (2 + 2 points)
Which one of you TAs committed those files?	3	B	3	- correct user name (3 points) - This is available if they used svn log
When did s/he commit them and what was the commit message?	4	S	4	- correct date/time for file1 (2 points) - correct date/time for file2 (2 points) -"Pins!"
6.1. Hello World (hello.c) (20 pts)	**Possible Points**	**Point Break Down**	**Points Awarded**	
hello.c produces no errors/warnings when compiled with	10	B	10	"gcc -Wall -Werror -o hello hello.c". (10 points)
hello.c emits only "Hello world!\n" when executed	3	S	3	partial credit for something "close".
hello.c returns with exit status 0 (or EXIT_SUCCESS)	2	B	2	"exit(0)" is OK.
Style (5 pts)	**Points Lost**	**Point Break Down**	**Deductions Awarded**	
Code should not declare function prototypes (e.g., "void exit(int)") instead of including a header (e.g., #include <stdlib.h>)	-2	B	0	
Code should not include unnecessary headers (e.g., string.h, etc)	-1	B	0	
"int main()" should receive comment that "int main(void)" is proper.	0	B	0	
"int main(void)", "int main(int argc, char *argv[])", "int main(int argc, char **argv)" are all OK, anything suspiciously different is not.	-2	B	0	
No code indentation.	-2	B	0	
Use your discretion for others.	-	B		
6.2. Goodbye World (goodbye.s) (20 pts)	**Possible Points**	**Point Break Down**	**Points Awarded**	
goodbye.s produces no errors/warnings when assembled with "gcc -o goodbye goodbye.s"	10	B	10	
goodbye.s emits only "Hello world!\nGoodbye world!\n" when executed	3	S	3	partial credit for something "close"
goodbye.s returns with exit status 42	2	B	2	
Style (5 pts)	**Points Lost**	**Point Break Down**	**Deductions Awarded**	
Code should not unnecessarily duplicate assembler directives / GCC metadata	-1	B	0	(i.e., only one instance of ".global main", ".type main, %function", ".size main, .-main", ".ident ...")
"Hello world!\n" and "Goodbye world!\n" strings should reside in the same section (.rodata is preferable, but .text is OK too). But one shouldn't reside in .text and the other in .rodata.	-1	B	0	Appending "Goodbye world!\n" to the end of "Hello world!\n" is a totally legitimate, full credit solution --it was not my intent, but there's no prohibition against it in the lab handout (there will be next year).
Use your discretion for others.	-	B	0	- Strings don't require any strict alignment, but it's OK if there are a few extra ".align" directives if they're sensible. - It is OK to duplicate ".section .rodata" or ".text" directives if the data that follows is appropriate, although it's probably a bit preferable to group all section contents together. - Use your discretion for others, but in general don't be too harsh on point deductions here because we really didn't expect them to be ARM assembly experts--we just wanted them to experiment around a little.
6.3. Disassembly (hello-d.txt, hello-D.txt, disassembly.txt) (20 pts)				
6.3.1. What is the entry point address of the program? (5 pts)	**Points Lost**	**Point Break Down**	**Deductions Awarded**	Should be "0x8348" or "0x8374" for nearly everyone. If they list another address, check "hello-d.txt" to see if the address for "_start" matches (which should correspond to the start of the .text section).
Address should correspond to "_start", others may look plausible (e.g., "_init", "main") but are wrong.	-2	B	0	
The answer should be an address (e.g., 0x8348) not a symbol (e.g., "_start")	-1	B	0	
Answers "0x8348" or "0x8374" need justification, although evidence provided in hello-d.txt is perfectly sufficient.	-4	B	0	In other words, if someone answers "0x8348" or "0x8374" and that address _does not_ correspond to "_start" in their hello-d.txt (or anything else plausible) likely "borrowed" the answer from another group. It's unlikely anyone will actually be marked off for this, but keep an eye out for it. Mark "no justification".
	Possible Points	**Point Break Down**	**Points Awarded**	
6.3.2. What is the name of the first function branched to in the program?	5	B	0	From "_start", the first branch is to address 0x8330 (if _start == 0x8348), which corresponds to the second entry of the PLT. However, looking at the symbol table or the relocation table reveals that the name of the function is "__uClibc_main" (just stating "__uClibc_main" is sufficient). Give full points if the answer is _start -- this turned out to be a confusing question for some reason.
6.3.3. What is the key difference between the output of objdump -d (hello-d.txt) and objdump -D (hello-D.txt)? (5 pts)	**Points Lost**	**Point Break Down**	**Deductions Awarded**	From objdump(1): -d Display the assembler mnemonics for theobjfile. This option only disassembles those sections which are machine instructions from expected to contain instructions. -D Like -d, but disassemble the contents of all sections, not just those expected to contain instructions. The answer should somehow convey that "-d" disassembles suspected instructions, while "-D" disassembles all data (whether legitimate instructions or not).
"key" functions vs "all" functions without specifying any actual distinction simply reiterates the question.	-3	B	0	

"-D" doesn't disassemble in "greater detail", it disassembles non-code sections	-2	B	0	
Use your discretion for others.	-	B	0	Answers which make some distinction between instruction (or instruction sections) and data (or data sections) generally should receive full credit. Major exceptions noted above.
	Possible Points	Point Break Down	Points Awarded	
6.3.4. Is the interpretation of the instructions under the .rodata section of hello-D.txt correct? What does this interpretation mean?	5	S	4	In short, the instructions in .rodata are meaningless. The actual data contained there is the ASCII representation of the string "Hello world!", and the instructions shown is just how the disassembler interprets the ASCII. While the instructions are worthless, the data itself isn't garbage--it's a string used in the program. - A totally correct (5/5) answer must recognize that the .rodata section: - Contains the string "Hello world!\n" (or at least, the string passed to the printf function). - Contains data was never meant to be interpreted as instructions (hence, explaining their odd nature). - Mentioning that the data was never meant to be interpreted as instructions, but not mentioning what the data actually is ("Hello world!\n" string) is worth - Any mention that .rodata contains a string is worth at least 3/5 points. Since the answers here are more subjective, I'll cite some examples: - "Yes, the RO data section deals with constant values in the program." While .rodata is where constants are stored, that's not why the interpretation is valid/invalid. This is just a general statement of what any .rodata section is, and contains no introspection. (-4 points) - "If you need to access data in your code that's static and won't be modified, you can save time and resources by processing the data once and caching it for later reuse. However, this doesn't make sense in helloworld because we are only referencing the data once." This answer doesn't make sense. (-4 points) Otherwise, use your best judgment and assign points on how much you feel the group "gets" the answer. It's probably worth taking a point or two off for a "fact" explained in the answer that's blatantly wrong, even if it's tangential.
7.5. Makefile Exercise (math.c, math.c, calc.c, Makefile). (20 points)	Possible Points	Point Break Down	Points Awarded	
Produces no errors/warnings when compiled with reference Makefile.	5	B	5	
Fully implements the proper execution behavior as specified in lab handout ("number operator number" input format, all five operations, signed decimal integer result, looping, invalid input termination)	5	S	2	- Note any verbose output, it's not part of the specification (-0 points). - Any missing component (any operator, loop, etc) subtracts 1 point.
example.mk is appropriately modified into the Makefile, should resemble the reference Makefile.	5	S	5	- Makefiles based on example.mk, but have been significantly modified (e.g., clobber target removed) should be commented with a warning that such changes are destructive and may affect grading scripts, etc. (-0 points) - Makefiles not based on example.mk (i.e., are much simpler, fully specify compile rules instead of using implicit rules, etc) did not follow the directions. This isn't a complex project, but later ones will be--so advanced Makefile features will be necessary. (-2 points) - Any Makefile that compiles the calc executable "properly" (with the right flags, etc.) even if it doesn't follow example.mk should still receive 3/5 points.
Style (5 pts)	Points Lost	Point Break Down	Deductions Awarded	
Any buffer overflow (gets, scanf with "% s") immediately gets marked "unsafe code".	-3	B	-3	- Using sscanf with appropriately sized buffers may be safe, but still shouldn't be encouraged--don't deduct points here, but see below.
Any "convolved" parsing (i.e., using sscanf("%s %s %s") to parse into strings, atoi() to parse strings into integers, etc) should deduct one or two points depending on degree of complexity:	-2	S	0	- In general, this program doesn't require sscanf("%s"), strtok, or any manual string parsing--simpler alternatives exist. - A one point deduction is probably reasonable in most cases, two if it's _really_ obtuse.
Naming the "div" function something else is fine since it doesn't conflict when including stdlib.h.	0		0	
No code indentation.	-2	S	0	Only -1 point if taken off previously
#including a .c file	-2	B	0	
Use your discretion for others.	-	B		
Handin Procedure Compliance (5 points)	Points Lost	Point Break Down	Deductions Awarded	The purpose of this lab was to familiarize students with the hardware platform, revisit Unix programming, learn what help resources are available, and understand the procedure for handins. Since this lab was not intended to be difficult, a significant portion of the points will be allotted to "following directions" including following the handin procedure.
Incorrect file names: file name compliance is helpful, if not essential for grading.	-2	S	0	- There's a separate deduction for Makefiles below, don't use this one. - There's a separate deduction for case conflicts, don't use this one. - Only take off a single point if "hello-d.txt" or "hello-D.txt" were renamed due to case conflicts. See below.
A "Makefile" won't work unless it's named "Makefile" (or "makefile", or "GNUMakefile")	-1	B	0	
Incorrect file name cases	-1	B	0	- There's no case sensitivity issue on Unix systems, "hello.c" and "HELLO.C" are distinct files and may coexist. Other operating systems may have issues with this, but it's the students' responsibility to rectify this before handin. - "hello.c" is C language source code, "hello.C" is C++ language source code--the compiler makes this distinction, and the resulting interpretation of code is different. - "goodbye.s" isn't preprocessed by cpp, "goodbye.S" is.
Incorrect directory hierarchy (i.e., flat submission directory)	-2 per file	B	-5	
Submitting a .tar.gz, .zip, or .rar file instead of individual source files.	-2	B	0	- Don't take points off if this was an email submission, since sending a tarball is the most sensible thing to do (sending a .rar file isn't).
Late submission	-10 points per day	Point Break Down	Deductions Awarded	
	-	B	0	

PART 2

Part 1:	Give at least	for Cost that is less than
Use gdb or calculate the cost using the table.	45	380
	46	375
Points Awarded:	47	370
50	48	365
Counted : 312	49	360
	50	355

(assuming everything else is fine and the optimized code does not produce a different output than un-optimized code).

Subtract 15 points if the result is not correct for the other test strings.

	# Instr	13
	Normal Instr	73
	Branch/Ld/Sts	58
	COUNT	312

Part 2:	Points Awarded

All of them should have realized that a simple way to determine oddball is to sum all the elements of the array and subtract this sum from n(n+1) where n is given below.
This code might look like
n=(len+1)/2
odd-val = n(n+1)-sum(arr[i]); //sum computed from 1 to len
or something similar.
 	
But this code has problems and is not equivalent to the un optimized code. The point to realize is that this kind of optimization will cause an overflow if n > 2^16 unless care is taken to prevent overflow. One simple way is to define
 	
long residual = n(n+1);
 	
for(i=0;i<len;i++)
residual -= arr[i];
oddball = (int) residual;
 	
since each element in the array was less than 2^(31)-1 (clarified on bboard), there will not be overflow. Give credit for other options of taking care of the overflow problem.
 	
subtract 10 points if the group did not even mention or take any steps to prevent overflow.
 	
The correct answer is by using xor's -- all the duplicate values will zero themselves out when computing the XORs leaving only the oddball value remaining in the residual.
 	
Since we asked for two versions -- if one of the versions is the XOR version don't be too picky about the other version as long as it does some optimization, give full credit. If one of the versions is the incorrect optimization I explained above, deduct points for not taking care of overflow but give full points for the other optimized version.
	40