Read me

First of all, I want to say good job to myself, there always gonna result great if I work hard enough. All right, I wouldn't be narcissism, let me explain what I did to make the program work.

All right, let me explain my understanding of the non-extra credit park which is emul. So the emul park will read a y86 file which contain a lot of y86 command like .long .byt etc. All of the text from the y86 have the address that points out where the instruction and statement will be saved in the memory. And the memory size will be given as the .size. Then save the y86 text with their own address in the memory. After I saved everything from y86 in my memory, I will find the location where .text begins. and read all the instruction and compare with the instruction of the assembly code and compile my code. Then I can return a phenomenal result from prog1.y86 and prog2.y86 . Now, Let me explain my code step by step.

First of all, I need to is to read the y86 file. I will open the file and read everything and save it in the char array. After I save anything in this array, I was kind afraid that I mess up this string so I create a copy function which will malloc the same length and save this array or this string in another char array. Then I can play with the copy part array, like I made the first array that copying from the file is unchangeable as a constant, but I can do anything I want for the second array. Every

time I finish using the second array I will free this array and if I will need it for next time, I will malloc again because I'm afraid if I did changed anything.

After read the file from give, the priority of the job is find out is the size, so that I will know how large the memory will and should be create, it will appear one time in the y86 file and one time only. So I will use a while loop to find if the size appeared twice and that will lead a Error. After I find out the size (convert to decimal) I will malloc a space for the memory as a unsigned char's array, which will have only one byte size for each block or location. Notice that 2 hex character will perform one byte, so I will combine two character and convert into Decimal and then convert it as unsigned character and save in the memory with their correspond address. I will use a while loop to do the read and save part. Notice that the .text will also appear once and one time only, which means if there's two .text appears in the instruction, the Error will be allocate. Also, I need to save the .text's address because when I try to compile my program with y86 code, I need to find where to start and .text's address will be where the program start.

After I read everything from the file and saved in the memory I created, I will now do the program compiler. I will not copy the Y86 instructions here but let me discuss with all these functions.

Y86	Y86	Assem	what will it do
0	0	nop	it will do nothing
1	0	halt	it will stop the program
2	0	rrmovl	The register copy to another register. it will ask for two register r1 and r2 and copy the thing from r1 to r2
3	0	irmovl	The immediate value copy to a register. it will ask for one register r1 32-bit value, then find the value and save in the register
4	0	rmmovl	The register copy to a memory block. it will ask for two register r1 and r2 and a displacement, then find the value from r2 and add the displacement which will result the address of the memory and copy everything from register1 to the memory block

Y86	Y86	Assem	what will it do
5	0	mrmovl	The memory block copy to a register. it will ask for two register r1 and r2 and a displacement, then find the value from r2 and add the displacement which will result the address of the memory and copy everything from that address of the memory to the register1
6	0-5	op(cal)	This will be “calculation” part, it will ask for two register and calculate and save the result in r2. And this part also include andl and xorl and compare.
7	0-6	jXX	This is the jump part, it could be conditional jump or non-conditional jump, the non-conditional part will be just simple because it will take a 32-bit destination and jump to that address, and the conditional jump will depend on the flag to see if that is suitable for jump
8	0	call	This one is really tricky, it is similar as jump also with a destination but different because it will push the return address to the memory.
9	0	ret	This one will pop out the return address from the memory and return
A	0	pushl	This part will push in the register to the memory
B	0	popl	This part will pop out the register to the memory
C	0-1	readX	it will simply read a 1 byte or 4 byte char
D	0-1	writeX	it will simply print out a 1 byte or 4 byte char
E	0	movsbl	it will ask for two register and one 32-bit displacement, it will move with sign extend from byte to longword.

Now I finished almost every part, and I want to clarify a bit more. For finding the register, because it was all saved in the memory and half byte to stand for the register corresponding number, so I will use the andl to help me to find the register and bit shift to find the exact number of the register. And the second part is when I have a 32 bit value and I wan to save it in my memory so I create a union and it contain an integer and a size 4 character array, so I will save the value in the array and find the corresponding decimal and monopoly the decimal.

So this is pretty much it, I love and don't love this program, it confused me a lot and and it worth 200 point. The assembly is confused enough now it’s y86, it’s the language more close to the machine instruction, but more far away readable. Sorry about the complain but this is also the

reason why I love this program, right now, the assembly code and y86 code make perfect sense to me and I get a sense of how the machine works. When I make this function it is tons of situations I need to consider both for read the file, save in the memory and for compile the instruction. It was tedious to debug a really small stupid mistake. But you cannot imagine how much relief I felt when I find out the mistake and how exciting and pound of myself when the program works perfectly fine.

"Because the people who are crazy enough to think they can change the world are the ones who do" ———— Steve Jobs