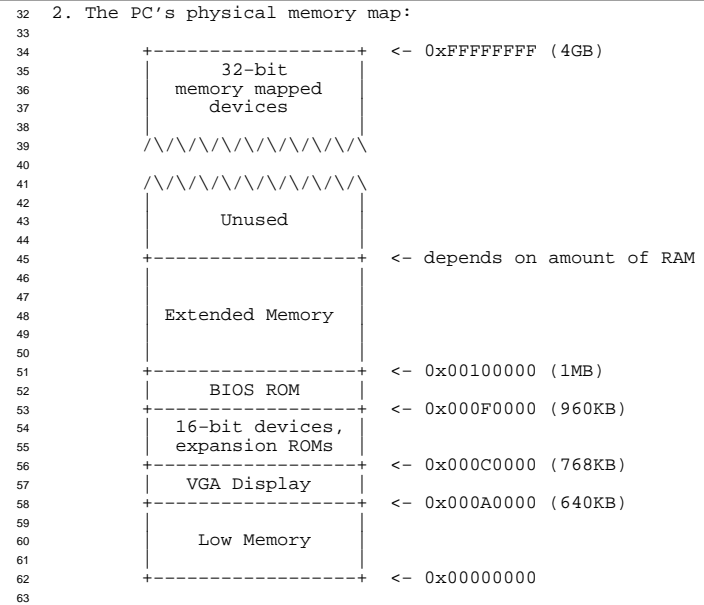


```

1 Handout for CS 439
2 Class 8
3 8 February 2013
4
5 [Credit to Frans Kaashoek, Robert Morris, and Nickolai Zeldovich.]
6
7 1. using the IN and OUT instructions
8
9     writing a byte to the parallel port (e.g., a line printer):
10
11     #define DATA_PORT    0x378
12     #define STATUS_PORT  0x379
13     #define BUSY         0x80
14     #define CONTROL_PORT 0x37A
15     #define STROBE       0x01
16
17     void
18     lpt_putc(int c)
19     {
20         /* wait for printer to consume previous byte */
21         while((inb(STATUS_PORT) & BUSY) == 0)
22             ;
23
24         /* put the byte on the parallel lines */
25         outb(DATA_PORT, c);
26
27         /* tell the printer to look at the data */
28         outb(CONTROL_PORT, STROBE);
29         outb(CONTROL_PORT, 0);
30     }
31

```



Feb 08, 13 12:15

I08-handout.txt

Page 3/4

```

64 3. Example
65
66 Here is the C code:
67 int main(void) { return f(8)+1; }
68 int f(int x) { return g(x); }
69 int g(int x) { return x+3; }
70
71 The assembly code:
72
73 _main:
74     prologue
75
76     pushl %ebp
77     movl %esp, %ebp
78
79     body
80     pushl $8
81     call _f
82     addl $1, %eax
83
84     epilogue
85     movl %ebp, %esp
86     popl %ebp
87     ret
88
89 _f:
90     prologue
91
92     pushl %ebp
93     movl %esp, %ebp
94
95     body
96     pushl 8(%esp)
97     call _g
98
99     epilogue
100    movl %ebp, %esp
101    popl %ebp
102    ret
103
104 <small version of _g>:
105    movl 4(%esp), %eax
106    addl $3, %eax
107    ret
108
109 <longer version of _g>:
110    prologue
111    pushl %ebp
112    movl %esp, %ebp
113
114    save %ebx
115    pushl %ebx
116
117    body
118    movl 8(%ebp), %ebx
119    addl $3, %ebx
120    movl %ebx, %eax
121
122    restore %ebx
123    popl %ebx
124
125    epilogue
126    movl %ebp, %esp
127    popl %ebp
128    ret
129

```

Feb 08, 13 12:15

I08-handout.txt

Page 4/4

```

130 4. Emulation of CPU in software
131
132 for (;;) {
133     read_instruction();
134     switch (decode_instruction_opcode()) {
135     case OPCODE_ADD:
136         int src = decode_src_reg();
137         int dst = decode_dst_reg();
138         regs[dst] = regs[dst] + regs[src];
139         break;
140     case OPCODE_SUB:
141         int src = decode_src_reg();
142         int dst = decode_dst_reg();
143         regs[dst] = regs[dst] - regs[src];
144         break;
145     ...
146     }
147     eip += instruction_length;
148 }
149
150
151 5. Emulate PC's physical memory map
152
153 #define KB      1024
154 #define MB      1024*1024
155
156 #define LOW_MEMORY  640*KB
157 #define EXT_MEMORY  10*MB
158
159 uint8_t low_mem[LOW_MEMORY];
160 uint8_t ext_mem[EXT_MEMORY];
161 uint8_t bios_rom[64*KB];
162
163
164 uint8_t read_byte(uint32_t phys_addr) {
165     if (phys_addr < LOW_MEMORY)
166         return low_mem[phys_addr];
167     else if (phys_addr >= 960*KB && phys_addr < 1*MB)
168         return bios_rom[phys_addr - 960*KB];
169     else if (phys_addr >= 1*MB && phys_addr < 1*MB+EXT_MEMORY) {
170         return ext_mem[phys_addr-1*MB];
171     } else ...
172 }
173
174 void write_byte(uint32_t phys_addr, uint8_t val) {
175     if (phys_addr < LOW_MEMORY)
176         low_mem[phys_addr] = val;
177     else if (phys_addr >= 960*KB && phys_addr < 1*MB)
178         /* ignore attempted write to ROM! */
179     else if (phys_addr >= 1*MB && phys_addr < 1*MB+EXT_MEMORY) {
180         ext_mem[phys_addr-1*MB] = val;
181     } else ...
182 }
183
184
185

```