Benchmark program:

SimpleAdd.c

```
1  int main() {
2
3      int a = 3;
4      int b = 2;
5      int c = 0;
6
7      c = a + b;
8
9      return c;
10 }
11
```

encoding of each instruction — convert this to binary

* pseudo instructions (encoding tells you what instr. they really are for this program)

⇒ look up top 6 bits in MIP manual Appendix Table A-2, if "special", look up last 6 bits in Table A-3.

Disassembly:

Disassembly of section .text:80020000 <main>:

look through list to identify which instructions are needed — look in MIPS manual for what it does to operands

```
80020000:  27bdffe8  addiu  sp,sp,-24    //1 (allocate stack space by moving stack pointer)
80020004:  afbe0014  sw     s8,20(sp)    //1 (backup the frame pointer)
80020008:  03a0f021  move*  s8,sp        //1 (move frame pointer to end of stack frame)
8002000c:  24020003  li*    v0,3         //3
80020010:  afc20000  sw     v0,0(s8)     //3  a = 3
80020014:  24020002  li     v0,2         //4
80020018:  afc20004  sw     v0,4(s8)     //4  b = 2
8002001c:  afc00008  sw     zero,8(s8)   //5  c = 0
80020020:  8fc30000  lw     v1,0(s8)     //7
80020024:  8fc20004  lw     v0,4(s8)     //7
80020028:  00621021  addu   v0,v1,v0     //7  a + b
8002002c:  afc20008  sw     v0,8(s8)     //7  c <--
80020030:  8fc20008  lw     v0,8(s8)     //9  return c (place it into 'v0', which is defined by MIPS as a return value register)
80020034:  03c0e821  move   sp,s8        // (undo the 3rd instr, reason not obvious here)
80020038:  8fbe0014  lw     s8,20(sp)    // (undo the 2nd instr, restore the frame pointer)
8002003c:  27bd0018  addiu  sp,sp,24     // (undo the 1st instr, deallocate stack space)
80020040:  03e00008  jr     ra           // jump to return address
80020044:  00000000  nop
```

don't worry about reg names, they are accessed by # in hardware, which is included in the instruction encoding