Example Program

Suppose array A[] = {5, 3, -6, 19, 8, 12}
is located in memory starting at location $0100

Add up all the numbers into register r1

```
add  r1, r0, r0
addi r5, r0, 0x0100      I'll write $0100 ← lazy way
                                or 0x0100 ← correct for Nios II
ldw  r2, 0(r5)      )
add  r1, r1, r2     )  5
ldw  r2, 4(r5)      )
add  r1, r1, r2     )  3
ldw  r2, 8(r5)      )
add  r1, r1, r2     )  -6
ldw  r2, 12(r5)     )
add  r1, r1, r2     )  19
ldw  r2, 16(r5)     )
add  r1, r1, r2     )  8
ldw  r2, 20(r5)     )
add  r1, r1, r2     )  12
```

A:

0x100	5
0x104	3
0x108	-6
0x10C	19
0x110	8
0x114	12

Note: this solution is very primitive, and it uses only the most basic features of an assembler (the ability to translate from text to binary instructions)

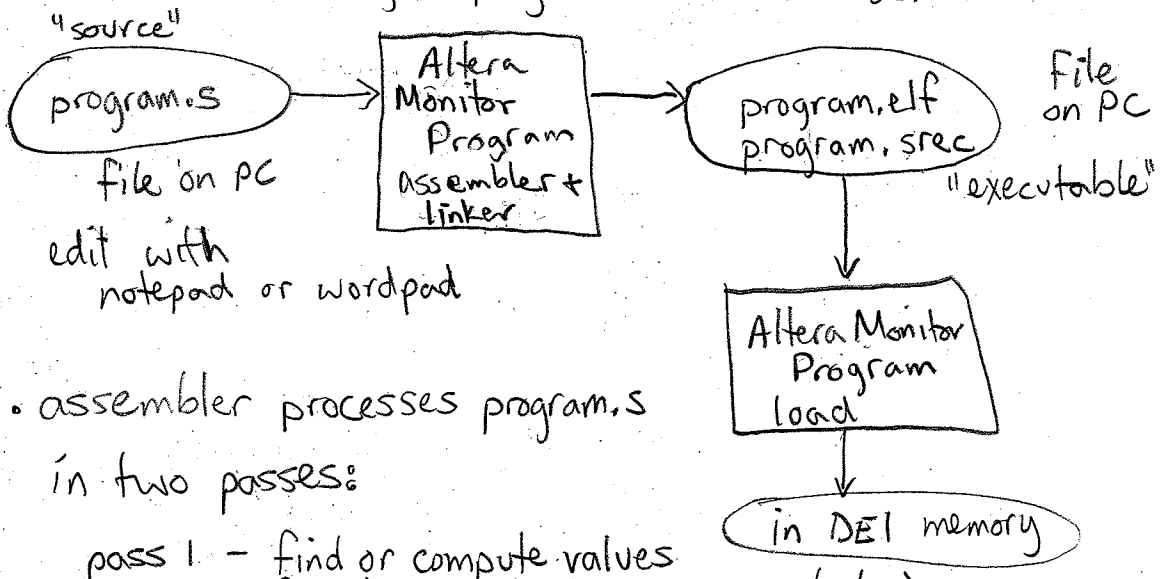
See solution 2 on handout for slightly better version using a loop.

See last page for an even better version that uses assembly language features.

Assembly Language Programming

- recall: each instruction is 32 bits
- examples
  - "addi r16, r16, -1" → $843F FFC4
  - "stw r17, 0(r16)" → $8440 0015
  - "stw r0, 4(r16)" → $8000 0115
- translating assembly language (write with text editor) → machine language (bits)

is done using a program called an "assembler"



- assembler processes program.s in two passes:
  - pass 1 - find or compute values for all symbols (always constants)
  - pass 2 - replace all symbols with binary value and output machine language bits in the executable
- Important: we will use [symbols (names)] to represent [constants (numbers)] or [constant expressions (numbers)]

& assembler manages them for us

- assembler lets us control how to organize instructions and data in memory using directives

directives always start with a dot, eg ".text"

- assembler also gives us "pseudoinstructions"
  - not real NIOS II instructions
  - they map easily to 1 or 2 real instructions

example:

pseudo - movi r3, 48 puts IMM16 into r3
real - addi r3, r0, 48 puts IMM16 into r3

pseudo - movia r3, 0x1234 5678 puts IMM32 into r3
real - orhi r3, r0, 0x1234
addi r3, r3, 0x5678
} breaks IMM32 into two IMM16

movi, movia are more clear and convenient

Labels and Loops

"br labelA" branch always to location "labelA"
ex: "STOP: br STOP"

"beq r3, r6, labelB" if(r3 == r6), go to labelB (otherwise, go to next instruction)

"bne _____" if(r3 != r6)

bgt _____ if(r3 > r6)

blt _____ if(r3 < r6)

bge _____ if(r3 ≥ r6)

ble _____ if(r3 ≤ r6)

```
/*
 * Example Nios II Programs
 *
 * An array A of words is located at 0x100.
 * Values of A[] = { 5, 3, -6, 19, 8, 12 },
 * so the memory contents are:
 *    0x0100    5
 *    0x0104    3
 *    0x0108   -6
 *    0x010C   19
 *    0x0110    8
 *    0x0114   12
 *
 * Goal: add up array values into register r1.
 */


/* There are two sample solutions below. They both
 * work, but they are both missing the array data.
 * Also, each program should be in a separate file.
 */

---------------------------------------------------------
/* Solution 1: Most basic version, uninteresting */

.global _start
_start:
      add   r1, r0, r0        /* init r1 to 0 */
      addi  r5, r0, 0x0100    /* init r5 to starting address of A */

      ldw   r2, 0(r5)         /* read 5 */
      add   r1, r1, r2        /* add 5 to r1 */
      ldw   r2, 4(r5)         /* read 3 */
      add   r1, r1, r2
      ldw   r2, 8(r5)         /* read -6 */
      add   r1, r1, r2
      ldw   r2, 12(r5)        /* read 19 */
      add   r1, r1, r2
      ldw   r2, 16(r5)        /* read 8 */
      add   r1, r1, r2
      ldw   r2, 20(r5)        /* read 12 */
      add   r1, r1, r2
STOP: br    STOP              /* infinite loop */

---------------------------------------------------------
/* Solution 2: Use a loop instead, adds label "LOOP" */

.global _start
_start:
      add   r1, r0, r0

      addi  r3, r0, 6         /* loop counter, 6 entries in A */
      addi  r5, r0, 0x0100    /* init r5 to starting address of A */

LOOP: ldw   r2, 0(r5)         /* read next entry in A[] */
      add   r1, r1, r2

      addi  r5, r5, 4         /* go to next entry in A */
      subi  r3, r3, 1         /* decrement loop counter */
      bne   r3, r0, LOOP      /* if r3!=0, go back to LOOP */
STOP: br    STOP              /* infinite loop */
```

```
/*
 * Example Nios II Program
 *
 * An array A of words is located at 0x100.
 * Values of A[] = { 5, 3, -6, 19, 8, 12 },
 * so the memory contents are:
 *    0x0100    5
 *    0x0104    3
 *    0x0108   -6
 *    0x010C   19
 *    0x0110    8
 *    0x0114   12
 *
 * Goal: add up array values into register r1.
 */

/* The solution below uses many assembly language features
 * to add memory layout, initial data values, symbols,
 * and pseudoinstructions to the program.
 */


/* The "constants" section comes first. This is where you put
 * all of your compiler directives such as ".equ" statements.
 * This section will not require any 'storage' in memory while
 * running your program; it is only a series of commands to the
 * assembler.
 */
.global _start
.equ N,  (Aend-Astart)/4      /* defines symbol N to hold constant 6 */

/* The ".text" section comes second and contains only program
 * instructions. The assembler creates a 32-bit word in memory
 * for every instruction.
 */
.text

_start:
      movi  r1, 0             /* initialize sum */
      movi  r3, N             /* loop counter, N entries in A */
      movia r5, Astart        /* Astart is address of A */

LOOP: ldw   r2, 0(r5)         /* read next entry in A[] */
      add   r1, r1, r2

      addi  r5, r5, 4         /* go to next entry in A */
      subi  r3, r3, 1         /* decrement loop counter */
      bne   r3, r0, LOOP      /* if r3!=0, go back to LOOP */

STOP: br    STOP

/* The ".data" section comes third and allocates memory for all variables
 * The assembler stores data in the same order it is given.
 */
.data

Astart:
.word 5, 3, -6, 19, 8, 12
Aend:

.end
```