

ECE 375 Lab 4

Large Number Arithmetic

Lab session: 015
Time: 12:00-13:50

Author: Astrid Delestine
Programming partner: Lucas Plastid

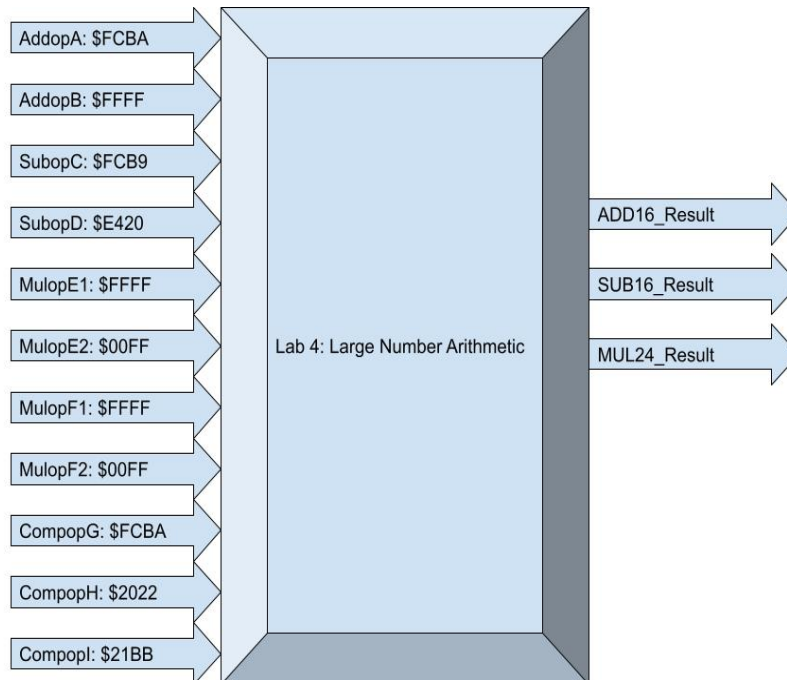
TA Signature

1 Introduction

This is the fourth lab in the ECE 375 series and it covers adding and subtracting words as well as multiplying words together. In this sense words designate 16 bit numbers. It is important to note that this assembly was written for the m128 chipset and not the regular atmega32u4 chipset. This is due to the fact that we can operate a simulation tool when using the m128 chipset that is not available when we use our regular chipset. The student will write their assembly to do these expected operations with large numbers and is given the expected inputs and can calculate the outputs.

2 Design

This lab Lucas and I collaborated and built out ideas for multi-byte addition, subtraction, and multiplication. It was quite interesting to see our ideas collide and how different approaches can be taken to the same problem. Addition can be considered very trivial, however subtraction becomes a little more difficult as the programmer has to consider negative values and if they are possible with this code. In the handout it clearly states that in this lab we will only be dealing with a more simple unsigned subtraction so the result can be held in two 8 bit registers. Next the multiplication of two 24 bit numbers. This will be more difficult than the previous two problems. In this case we decided to solve the problem linearly and not worry about looping or recursion. Finally we move on to the compound function, which uses each of the previously implemented methods to solve the problem of $((G - H) + I)^2$.



3 Assembly Overview

As for the Assembly program an overview can be seen below

3.1 Internal Register Definitions and Constants

The multipurpose register was setup as r16. At r0 and r1 any multiplication output will be set, such that the outputs of any multiplication operation are automatically assigned to them. A default zero register is set to r2. Two other generic variable registers are defined as r3 and r4. Finally two registers named oloop and iloop are used for counting within the assembly itself.

3.2 Initialization Routine

Firstly the stack pointer is initialized, then the register defined above as the zero register is cleared.

3.3 Main Routine

The main operations that happen in the main routine are those that are initialized below and are called in this order. Firstly the adding operations take place, those being LOADADD16 and ADD16. Next the functions associated with the subtraction operation are called, those being LOADSUB16 and SUB16. Next the functions referencing the MUL24 operation are called. In order they are called LOADMUL24 and MUL24. Finally the compound function set is called, being LOADCOMPOUND and COMPOUND. Once all of these functions have been called the main method loops at the done flag, determining the program to be complete.

3.4 Subroutines

3.5 ADD16

This subroutine adds two

3.6 SUB16

3.6.1 MUL24

3.6.2 ADDMUL2x

3.6.3 LOADMUL24

3.6.4 LOADADD16

3.6.5 LOADSUB16

3.6.6 COMPOUND

3.6.7 LOADCOMPOUND

3.6.8 CLRRES

3.6.9 MUL16

3.6.10 FUNC

4 Testing

Tested each button

Case	Expected	Actual meet expected
D4 Pressed	Clears the Display	✓
D5 Pressed	Shows 2 lines of text, Each name	✓
D6 Held after D7 is Pressed	Cancels Marquee	✓
D7 Pressed	Begins Marquee	✓

Table 1: Assembly Testing Cases

5 Study Questions

1. Although we dealt with unsigned numbers in this lab, the ATmega32 micro-controller also has some features which are important for performing signed arithmetic. What does the V flag in the status register indicate? Give an example (in binary) of two 8-bit values that will cause the V flag to be set when they are added together.
2. In the skeleton file for this lab, the .BYTE directive was used to allocate some data memory locations for MUL16's input operands and result. What are some benefits of using this directive to organize your data memory, rather than just declaring some address constants using the .EQU directive?
3. In computing, there are traditionally two ways for a microprocessor to listen to other devices and communicate: polling and interrupts. Give a concise overview/description of each method, and give a few examples of situations where you would want to choose one method over the other.

4. Describe the function of each bit in the following ATmega32U4 I/O registers: EICRA, EICRB, and EIMSK. Do not just give a brief summary of these registers; give specific details for each bit of each register, such as its possible values and what function or setting results from each of those values. Also, do not just directly paste your answer from the datasheet, but instead try to describe these details in your own words.
5. The ATmega32U4 microcontroller uses interrupt vectors to execute particular instructions when an interrupt occurs. What is an interrupt vector? List the interrupt vector (address) for each of the following ATmega32U4 interrupts: Timer/Counter0 Overflow, External Interrupt 6, and Analog Comparator.
6. Microcontrollers often provide several different ways of configuring interrupt triggering, such as level detection and edge detection. Suppose the signal shown in Figure 1 was connected to a microcontroller pin that was configured as an input and had the ability to trigger an interrupt based on certain signal conditions. List the cycles (or range of cycles) for which an external interrupt would be triggered if that pin's sense control was configured for: (a) rising edge detection, (b) falling edge detection, (c) low level detection, and (d) high level detection. Note: There should be no overlap in your answers, i.e., only one type of interrupt condition can be detected during a given cycle.

6 Difficulties

This lab was not too difficult, however determining exactly how the marquee needed to work was definitely a challenge. After sorting out the bugs, it was extremely exciting to see text show up on the display.

7 Conclusion

This lab really helped teach just how peripherals can work, and how the ATMEGA32U4 handles certain peripherals. Several parts were challenging however these stimulating moments allowed the student to learn and understand what they needed to do at the same time.

8 Source Code

Listing 1: Assembly Bump Bot Script

```

1  ;*****
2  ;*   This is the skeleton file for Lab 4 of ECE 375
3  ;*
4  ;*   Author: Astrid Delestine and Lucas Plastied
5  ;*   Date: 2/16/2023
6  ;*
7  ;*****
8
9  .include "m128def.inc"           ; Include definition file

```

```

10
11 ;*****
12 ;*   Internal Register Definitions and Constants
13 ;*****
14 .def      mpr = r16          ; Multipurpose register
15 .def      rlo = r0          ; Low byte of MUL result
16 .def      rhi = r1          ; High byte of MUL result
17 .def      zero = r2         ; Zero register, set to zero in INIT,
18                               ; useful for calculations
19 .def      A = r3            ; A variable
20 .def      B = r4            ; Another variable
21
22 .def      oloop = r17        ; Outer Loop Counter
23 .def      iloop = r18        ; Inner Loop Counter
24
25
26 ;*****
27 ;*   Start of Code Segment
28 ;*****
29 .cseg                        ; Beginning of code segment
30
31 ;-----
32 ; Interrupt Vectors
33 ;-----
34 .org      $0000              ; Beginning of IVs
35          rjmp      INIT      ; Reset interrupt
36
37 .org      $0056              ; End of Interrupt Vectors
38
39 ;-----
40 ; Program Initialization
41 ;-----
42 INIT:                          ; The initialization routine
43
44          ; Initialize Stack Pointer
45          ldi        mpr, low(RAMEND)
46          out        SPL, mpr
47          ldi        mpr, high(RAMEND)
48          out        SPH, mpr
49          ; TODO
50
51          clr        zero      ; Set the zero register to zero,
52                               ; maintain these semantics, meaning,
53                               ; don't load anything else into it.
54
55 ;-----

```

```

56 ; Main Program
57 ;-----
58 MAIN:                                ; The Main program
59
60     ; Call function to load ADD16 operands
61     rcall LOADADD16
62     ; Operands stored in $0110 and $0112
63     nop ; Check load ADD16 operands (Set Break point here #1)
64     rcall ADD16
65     ; Call ADD16 function to display its results
66     ;(calculate FCBA + FFFF)
67     ; Result stored in $0120, should be $1FCB9
68     nop ; Check ADD16 result (Set Break point here #2)
69
70
71     ; Call function to load SUB16 operands
72     rcall LOADSUB16
73     ; Operands stored in $0114 and $0116
74     nop ; Check load SUB16 operands (Set Break point here #3)
75
76     ; Call SUB16 function to display its results
77     ;(calculate FCB9 - E420)
78     rcall SUB16
79     ; Result stored in $0130, should be $1899
80     nop ; Check SUB16 result (Set Break point here #4)
81
82
83     ; Call function to load MUL24 operands
84     rcall LOADMUL24
85     ; Operands stored in $0118 and $011B
86     nop ; Check load MUL24 operands (Set Break point here #5)
87
88     ; Call MUL24 function to display its results
89     ;(calculate FFFFFFFF * FFFFFFFF)
90     rcall MUL24
91     ; Result stored in $0140, should be $FFFFFFE000001
92     nop ; Check MUL24 result (Set Break point here #6)
93
94     ; Setup the COMPOUND function direct test
95     rcall LOADCOMPOUND
96     ; Operands stored in
97     ;$0114 (G = $FCBA),
98     ;$0116 (H = $2022), and
99     ;$0112 (I = $21BB)
100    nop ; Check load COMPOUND operands (Set Break point here #7)
101

```

```

102      ; Call the COMPOUND function , ((G-H)+I)^2
103      rcall COMPOUND
104      ; Result stored in $0140, should be $0000FCA8CEE9
105      nop ; Check COMPOUND result (Set Break point here #8)
106
107 DONE:    rjmp     DONE                ; Create an infinite while
108                                         ; loop to signify the
109                                         ; end of the program.
110
111 ; *****
112 ;*  Functions and Subroutines
113 ; *****
114
115 ;-----
116 ; Func: ADD16
117 ; Desc: Adds two 16-bit numbers and generates a 24-bit number
118 ;       where the high byte of the result contains the carry
119 ;       out bit.
120 ;-----
121 ADD16:
122      push mpr
123      push A
124      push XH
125      push YH
126      push ZH
127      push XL
128      push YL
129      push ZL
130
131      clr mpr
132      bclr 0 ; CLEAR THE CARRY FLAG!!! (just in case lamp)
133      ; Load beginning address of first operand into X
134      ldi     XL, low(ADD16_OP1) ; Load low byte of address
135      ldi     XH, high(ADD16_OP1) ; Load high byte of address
136      ; Load beginning address of second operand into Y
137      ldi     YL, low(ADD16_OP2) ; Load low byte of address
138      ldi     YH, high(ADD16_OP2) ; Load high byte of address
139      ; Load beginning address of result into Z
140      ldi     ZL, low(ADD16_Result) ; points the end of Z
141      ldi     ZH, high(ADD16_Result)
142      ; Execute the function
143      ; 2 16 bit numbers being added generates a max of 24 bit number
144      ; ie 1111_1111_1111_1111
145      ; +1111_1111_1111_1111
146      ; -----
147      ; 1_1111_1111_1111_1110

```



```

148         ; can do 8 bits at a time
149
150
151
152         ; add 2 lowest bytes
153         ld A, X+
154         add mpr, A
155         ld A, Y+
156         add mpr, A
157         ; mpr now equals x + y & carry flag is included
158         ; store this lower result in the lowest memory of the result
159         st Z+, mpr
160         clr mpr
161
162         ld A, X+
163         adc mpr, A
164         ld A, Y+
165         add mpr, A
166         ; mpr now equals x + y & carry flag is included
167         ; store this lower result in the lowest memory of the result
168         st Z+, mpr
169         clr mpr
170
171
172         ; if c flag is set
173         brc noCarry;
174         ldi mpr, $01;
175         st Z, mpr
176 noCarry:
177         pop ZL
178         pop YL
179         pop XL
180         pop ZH
181         pop YH
182         pop XH
183         pop A
184         pop mpr
185         ret                                ; End a function with RET
186
187 ;-----
188 ; Func: SUB16
189 ; Desc: Subtracts two 16-bit numbers and generates a 16-bit
190 ; result. Always subtracts from the bigger values.
191 ;-----
192 SUB16:
193         ; Execute the function here

```

```

194      push mpr
195      push A
196      push B
197      push XH
198      push YH
199      push ZH
200      push XL
201      push YL
202      push ZL
203
204      ; Load beginning address of first operand into X
205      ldi      XL, low(SUB16_OP1) ; Load low byte of address
206      ldi      XH, high(SUB16_OP1) ; Load high byte of address
207      ; Load beginning address of second operand into Y
208      ldi      YL, low(SUB16_OP2) ; Load low byte of address
209      ldi      YH, high(SUB16_OP2) ; Load high byte of address
210      ; Load beginning address of result into Z
211      ldi      ZL, low(SUB16_Result) ; points the end of Z
212      ldi      ZH, high(SUB16_Result)
213
214      ld A, X+      ; Load low byte of OP1 into A,
215                    ; X now points at high byte
216      ld B, Y+      ; Load low byte of OP2 into B,
217                    ; Y now points at high byte
218      sub A, B      ; Subtract low byte of OP2 from low byte of OP1
219      st Z+, A      ; Store result into low byte of SUB16_Result,
220                    ; Z now points to high byte
221      ld A, X      ; Load high byte of OP1 into A
222      ld B, Y      ; Load high byte of OP2 into B
223      sbc A, B      ; Subtract high byte of OP2 from
224                    ; low byte of OP1 with carry
225
226      st Z, A      ; Store result to high byte of SUB16_Result
227
228      pop ZL
229      pop YL
230      pop XL
231      pop ZH
232      pop YH
233      pop XH
234      pop B
235      pop A
236      pop mpr
237      ret          ; End a function with RET
238
239 ;

```

```

240 ; Func: MUL24
241 ; Desc: Multiplies two 24-bit numbers and generates a 48-bit
242 ;       result.
243 ;-----
244 MUL24:
245 ; Simply adopting MUL16 ideas to MUL24 will not give you steady results
246 ; You should come up with different ideas.
247 /*
248 Imagine we are multiplying two 24-bit numbers, ABC and DEF.
249 A is the highest byte, C is the lowest byte. So A is referring
250 to only the highest byte of ABC. Imagine that when multiplying
251 C and F, the result is CFH:CFL where CFH and CFL are the high
252 and low bytes respectively of the result of multiplying C and F.
253
254 If you were to write out
255 how to multiply these together with individual 8-bit multiplication,
256 it might look something like this:
257
258           A   B   C   *
259           D   E   F
260 -----
261                   CFH CFL
262                   CEH CEL
263                   CDH CDL
264                   BFH BFL
265                   BEH BEL
266                   BDH BDL
267                   AFH AFL
268                   AEH AEL
269 +   ADH ADL
270   5   4   3   2   1   0
271 -----
272
273 This result does in fact take up 6 bytes, or 48-bits.
274 Something to keep in mind here is that when adding each
275 individual result to the total (such as CDL) there will
276 be previous results already added to that byte. That means
277 that we need to keep in mind carries. There will be no
278 carry coming out of the last byte, as two 24-bit numbers
279 multiplied together (FFFFFF x FFFFFFF) have a maximum
280 possible value of FFFFFFFE000001, which is only 6 bytes!
281
282 The way I will handle carries is every time I do addition,
283 I will check the carry bit to see if it is set. If it is,
284 I will move up where I am looking at by one byte, then add
285 the carry. Since this is also addition, I will check the

```

```

286 carry AGAIN! Repeat until no more carries. This means
287 I could carry up to 4 times, (the lowest byte will never
288 carry since it is only added to once) so for reliabilities
289 sake I will use a loop to check/add carries instead of just
290 adding the carry to every possibly byte even if there is
291 no carry.
292
293 */
294     ; Execute the function here
295     push XH      ; Push literally everything because
296     push XL      ; I have no idea what I will need :)
297     push YH
298     push YL
299     push ZH
300     push ZL
301     push mpr
302     push rlo
303     push rhi
304     push A
305     push B
306     push iloop
307     push oloop
308
309     ; Load beginning address of MUL24 result to X
310     ldi XL, low(MUL24.Result)
311     ldi XH, high(MUL24.Result)
312     st X+, zero
313     st X+, zero
314     st X+, zero
315     st X+, zero
316     st X+, zero
317     st X, zero ; Clear result just in case!
318
319     ; Load beginning address of first operand into Z
320     ldi ZL, low(MUL24.OP1) ; Load low byte of address
321     ldi ZH, high(MUL24.OP1) ; Load high byte of address
322     ; Load beginning address of second operand into Y
323     ldi YL, low(MUL24.OP2) ; Load low byte of address
324     ldi YH, high(MUL24.OP2) ; Load high byte of address
325     ; Load beginning address of result into X
326     ldi XL, low(MUL24.Result) ; points the end of X
327     ldi XH, high(MUL24.Result)
328
329 /*
330     A   B   C   *           (Z pointer)
331     D   E   F           (Y pointer)

```

332			2	1	0		offset
333	<hr/>						
334						CF	
335						CE	
336				CD			
337					BF		
338				BE			
339			BD				
340				AF			
341			AE				
342		AD					
343	5	4	3	2	1	0	(X offset)
344	<hr/>						
345	*/						
346							
347	ld A, Z	; Load C byte of OP1 to A					
348		; A will hold the first op					
349	ld B, Y	; Load F byte of OP1 to B					
350		; B will hold the second op					
351	mul A, B	; C*F					
352	r call ADDMUL2X	; Add to result					
353	;	A	B	C	*	(Z pointer)	
354	;	D	E	F		(Y pointer)	
355	;	2	1	0		offset	
356	adiw XH:XL, 1	; X offset = 1, changed so that ADDMUL2X					
357		; adds to the correct place					
358		; Need to do CE and BF					
359	;	A	B	C	*	(Z pointer)	
360	;	D	E	F		(Y pointer)	
361	;	2	1	0		offset	
362	ld A, Z	; A ← C					
363	ldd B, Y+1	; B ← E					
364	mul A, B	; C*E					
365	r call ADDMUL2X	; Add to result					
366	ldd A, Z+1	; A ← B					
367	ld B, Y	; B ← F					
368	mul A, B	; B*F					
369	r call ADDMUL2X						
370	adiw XH:XL, 1	; X offset = 2					
371		; Need to do CD, BE, and AF					
372	;	A	B	C	*	(Z pointer)	
373	;	D	E	F		(Y pointer)	
374	;	2	1	0		offset	
375	ld A, Z	; A ← C					
376	ldd B, Y+2	; B ← D					
377	mul A, B	; C*D					

```

378      rcall ADDMUL2X
379      ldd A, Z+1  ; A ← B
380      ldd B, Y+1  ; B ← E
381      mul A, B    ; B * E
382      rcall ADDMUL2X
383      ldd A, Z+2  ; A ← -A
384      ld B, Y     ; B ← -F
385      mul A, B    ; A * F
386      rcall ADDMUL2X
387      adiw XH:XL, 1 ; X offset = 3
388                        ; Need to do BD and AE
389      ;           A   B   C   *           (Z pointer)
390      ;           D   E   F           (Y pointer)
391      ;           2   1   0           offset
392      ldd A, Z+1  ; A ← B
393      ldd B, Y+2  ; B ← D
394      mul A, B    ; B * D
395      rcall ADDMUL2X
396      ldd A, Z+2  ; A ← A (nice)
397      ldd B, Y+1  ; B ← E
398      mul A, B    ; A * E
399      rcall ADDMUL2X
400      adiw XH:XL, 1 ; X offset = 4
401                        ; Need to do AD
402      ;           A   B   C   *           (Z pointer)
403      ;           D   E   F           (Y pointer)
404      ;           2   1   0           offset
405      ldd A, Z+2  ; A ← A (nice)
406      ldd B, Y+2  ; B ← D
407      mul A, B    ; A * D
408      rcall ADDMUL2X
409      ; done!
410
411      pop oloop
412      pop iloop
413      pop B
414      pop A
415      pop rhi
416      pop rlo
417      pop mpr
418      pop ZL
419      pop ZH
420      pop YL
421      pop YH
422      pop XL
423      pop XH

```

```

424
425         ret                                ; End a function with RET
426
427 ;-----
428 ; Func: ADDMUL2X
429 ; Desc: Adds a partial multiplication result word to X, assuming
430 ; that X is already pointing to where the low result
431 ; byte should be placed
432 ; AKA if result of the multiplication needs to be placed
433 ; into the X starting at the second byte, then X had
434 ; better already be pointing to the second byte.
435 ; Multiplication should already be done and sitting in
436 ; rlo and rhi!
437 ;-----
438 ADDMUL2X:
439     push XL
440     push XH
441     push A
442
443     ld A, X      ; Pull out low byte from X to A
444     add A, rlo   ; Add rlo to A
445     st X+, A     ; Place A back into X, inc X
446     ld A, X      ; Pull out high byte from X to A
447     adc A, rhi   ; Add rhi to high byte plus carry
448     st X+, A     ; Place A back into X, inc X
449     ; Carry until no longer carry
450 addmulloop:
451     brcc addmulfinish ; Finish if carry no longer set
452     ld A, X      ; Pull out current byte from X
453     adc A, zero   ; Add carry to current byte
454     st X+, A     ; Place back into X, inc X
455     rjmp addmulloop ; Return to begining of loop
456 addmulfinish:
457
458     pop A
459     pop XH
460     pop XL
461     ret
462
463 ;-----
464 ; Func: LOADMUL24
465 ; Desc: Loads the numbers needed for the example MUL24
466 ;-----
467 LOADMUL24:
468     ; Execute the function here
469     push YH ; push regs to stack

```

```

470      push YL
471      push ZH
472      push ZL
473      push mpr
474      push iloop
475      push oloop
476
477      ; Uses OperandE1, OperandE2, OperandF1, and OperandF2
478      ; Placing these into MUL24_OP1 and MUL24_OP2 respectively
479      ldi ZH, high(OperandE1) ; load OperandE1 location to Z
480      ldi ZL, low(OperandE1)
481      ; Shift Z to prepare for program memory access:
482      lsl ZH
483      lsl ZL
484      adc ZH, zero ; shift carry from lower byte to upper byte
485      ldi YH, high(MUL24_OP1) ; Load OP1 location into Y
486      ldi YL, low(MUL24_OP1) ; ($0118)
487
488      ldi oloop, 3 ; load oloop with 3 to loop 3 times.
489 mulloadloop1:
490      lpm mpr, Z+ ; load mpr from Z, inc Z
491      st Y+, mpr ; store mpr to Y, inc Y
492      dec oloop ; decrement oloop to run loop 3 times
493      brne mulloadloop1
494      ; since operand E2 is immediately after E1 in the program data
495      ; we should be able to just increment to it :)
496      ; Operand E is now loaded to MUL24_OP1
497
498      ldi ZH, high(OperandF1) ; load OperandF1 location to Z
499      ldi ZL, low(OperandF1)
500      ; Shift Z to prepare for program memory access:
501      lsl ZH
502      lsl ZL
503      adc ZH, zero ; shift carry from lower byte to upper byte
504      ldi YH, high(MUL24_OP2) ; Load OP1 location into Y
505      ldi YL, low(MUL24_OP2) ; ($011B)
506
507      ldi oloop, 3 ; load oloop with 3 to loop 3 times.
508 mulloadloop2:
509      lpm mpr, Z+ ; load mpr from Z, inc Z
510      st Y+, mpr ; store mpr to Y, inc Y
511      dec oloop ; decrement oloop to run loop 3 times
512      brne mulloadloop2
513      ; Both operands should be loaded into program mem now!
514
515      pop oloop ; pop regs from stack

```



```

516      pop  iloop
517      pop  mpr
518      pop  ZL
519      pop  ZH
520      pop  YL
521      pop  YH
522
523      ret                                ; End a function with RET
524
525 ;-----
526 ; Func: LOADADD16
527 ; Desc: Loads the numbers needed for the example ADD16
528 ;-----
529 LOADADD16:
530      ; Execute the function here
531      push YH ; push regs to stack
532      push YL
533      push ZH
534      push ZL
535      push mpr
536      push iloop
537      push oloop
538
539      ; Uses OperandA and OperandB
540      ; Placing these into ADD16_OP1 and ADD16_OP2 respectively
541      ldi ZH, high(OperandA) ; load OperandA location to Z
542      ldi ZL, low(OperandA)
543      ; Shift Z to prepare for program memory access:
544      lsl ZH
545      lsl ZL
546      adc ZH, zero ; shift carry from lower byte to upper byte
547      ldi YH, high(ADD16_OP1) ; Load OP1 location into Y
548      ldi YL, low(ADD16_OP1) ; ($0110)
549
550      ldi oloop, 2 ; load oloop with 2 to loop 2 times.
551 addloadloop1:
552      lpm mpr, Z+ ; load mpr from Z, inc Z
553      st Y+, mpr ; store mpr to Y, inc Y
554      dec oloop ; decrement oloop to run loop 2 times
555      brne addloadloop1
556      ; Operand A is now loaded to ADD16_OP1
557
558      ldi ZH, high(OperandB) ; load OperandB location to Z
559      ldi ZL, low(OperandB)
560      ; Shift Z to prepare for program memory access:
561      lsl ZH

```

```

562     lsl ZL
563     adc ZH, zero ; shift carry from lower byte to upper byte
564     ldi YH, high(ADD16_OP2) ; Load OP2 location into Y
565     ldi YL, low(ADD16_OP2) ; ($0112)
566
567     ldi oloop, 2 ; load oloop with 2 to loop 2 times.
568 addloadloop2:
569     lpm mpr, Z+ ; load mpr from Z, inc Z
570     st Y+, mpr ; store mpr to Y, inc Y
571     dec oloop ; decrement oloop to run loop 2 times
572     brne addloadloop2
573     ; Both operands should be loaded into program mem now!
574
575     pop oloop ; pop regs from stack
576     pop iloop
577     pop mpr
578     pop ZL
579     pop ZH
580     pop YL
581     pop YH
582
583     ret ; End a function with RET
584
585 ;-----
586 ; Func: LOADSUB16
587 ; Desc: Loads the numbers needed for the example SUB16
588 ;-----
589 LOADSUB16:
590     ; Execute the function here
591     push YH ; push regs to stack
592     push YL
593     push ZH
594     push ZL
595     push mpr
596     push iloop
597     push oloop
598
599     ; Uses OperandC and OperandD
600     ; Placing these into SUB16_OP1 and SUB16_OP2 respectively
601     ldi ZH, high(OperandC) ; load OperandC location to Z
602     ldi ZL, low(OperandC)
603     ; Shift Z to prepare for program memory access:
604     lsl ZH
605     lsl ZL
606     adc ZH, zero ; shift carry from lower byte to upper byte
607     ldi YH, high(SUB16_OP1) ; Load OP1 location into Y

```

```

608         ldi YL, low(SUB16_OP1)    ; ($0114)
609
610         ldi oloop, 2              ; load oloop with 2 to loop 2 times.
611 subloadloop1:
612         lpm mpr, Z+ ; load mpr from Z, inc Z
613         st Y+, mpr ; store mpr to Y, inc Y
614         dec oloop                ; decrement oloop to run loop 2 times
615         brne subloadloop1
616         ; Operand C is now loaded to ADD16_OP1
617
618         ldi ZH, high(OperandD)    ; load OperandD location to Z
619         ldi ZL, low(OperandD)
620         ; Shift Z to prepare for program memory access:
621         lsl ZH
622         lsl ZL
623         adc ZH, zero ; shift carry from lower byte to upper byte
624         ldi YH, high(SUB16_OP2) ; Load OP2 location into Y
625         ldi YL, low(SUB16_OP2)    ; ($0116)
626
627         ldi oloop, 2              ; load oloop with 2 to loop 2 times.
628 subloadloop2:
629         lpm mpr, Z+ ; load mpr from Z, inc Z
630         st Y+, mpr ; store mpr to Y, inc Y
631         dec oloop                ; decrement oloop to run loop 2 times
632         brne subloadloop2
633         ; Both operands should be loaded into program mem now!
634
635         pop oloop                ; pop regs from stack
636         pop iloop
637         pop mpr
638         pop ZL
639         pop ZH
640         pop YL
641         pop YH
642
643         ret                      ; End a function with RET
644
645 ;-----
646 ; Func: COMPOUND
647 ; Desc: Computes the compound expression  $((G - H) + I)^2$ 
648 ;       by making use of SUB16, ADD16, and MUL24.
649 ;
650 ;       D, E, and F are declared in program memory, and must
651 ;       be moved into data memory for use as input operands.
652 ;
653 ;       All result bytes should be cleared before beginning.

```

```

654 ;-----
655 COMPOUND:
656
657     ; Setup SUB16 with operands G and H
658     ; Already done in LOADCOMPOUND
659     ; Perform subtraction to calculate G - H:
660     rcall SUB16
661     ; Setup the ADD16 function with SUB16 result and operand I
662     ; Operand I already loaded to ADD16_OP2 by LOADCOMPOUND
663     ; Just load SUB16_Result into ADD16_OP1:
664     ldi XL, low(SUB16_Result)
665     ldi XH, high(SUB16_Result)
666     ldi YL, low(ADD16_OP1)
667     ldi YH, high(ADD16_OP1)
668     ld mpr, X+
669     st Y+, mpr
670     ld mpr, X
671     st Y, mpr
672     ; Perform addition next to calculate (G - H) + I:
673     rcall ADD16
674     ; Setup the MUL24 function with ADD16 result as both operands:
675     ldi XL, low(ADD16_Result)
676     ldi XH, high(ADD16_Result)
677     ldi YL, low(MUL24_OP1)
678     ldi YH, high(MUL24_OP1)
679     ldi ZL, low(MUL24_OP2)
680     ldi ZH, high(MUL24_OP2)
681     ld mpr, X+
682     st Y+, mpr
683     st Z+, mpr
684     ld mpr, X+
685     st Y+, mpr
686     st Z+, mpr
687     ld mpr, X
688     st Y, mpr
689     st Z, mpr
690     ; Perform multiplication to calculate ((G - H) + I)^2:
691     rcall MUL24
692     ret                                     ; End a function with RET
693 ;-----
694 ; Func: LOADCOMPOUND
695 ; Desc: Loads the numbers needed for the compound, as well
696 ;       as clearing the result locations from previous
697 ;       functions first.
698 ;-----
699 LOADCOMPOUND:

```

```

700      ; Execute the function here
701      push YH ; push regs to stack
702      push YL
703      push ZH
704      push ZL
705      push mpr
706      push iloop
707      push oloop
708
709      rcall CLRRES ; Clear result memory locations
710
711      ; Uses OperandG, OperandH, and OperandI
712      ; as ( ( G - H ) + I )^2
713      ; Meaning SUB16 with G and H
714      ; Then ADD16 with the result and I
715      ; Then MUL24 where both operands are the result
716      ; So G and H need to be loaded to
717      ; SUB16_OP1 and SUB16_OP2 respectively
718      ; "I" will be loaded to ADD16_OP2 due to where
719      ; it visually is in the equation,
720      ; although it doesn't matter too much
721
722      ldi ZH, high(OperandG) ; load OperandG location to Z
723      ldi ZL, low(OperandG)
724      ; Shift Z to prepare for program memory access:
725      lsl ZH
726      lsl ZL
727      adc ZH, zero ; shift carry from lower byte to upper byte
728      ldi YH, high(SUB16_OP1) ; Load OP1 location into Y
729      ldi YL, low(SUB16_OP1) ; ($0114)
730
731      ldi oloop, 2 ; load oloop with 2 to loop 2 times.
732      comploadloop1:
733      lpm mpr, Z+ ; load mpr from Z, inc Z
734      st Y+, mpr ; store mpr to Y, inc Y
735      dec oloop ; decrement oloop to run loop 2 times
736      brne comploadloop1
737      ; Operand G is now loaded to SUB16_OP1
738
739      ldi ZH, high(OperandH) ; load OperandD location to Z
740      ldi ZL, low(OperandH)
741      ; Shift Z to prepare for program memory access:
742      lsl ZH
743      lsl ZL
744      adc ZH, zero ; shift carry from lower byte to upper byte
745      ldi YH, high(SUB16_OP2) ; Load OP2 location into Y

```

```

746      ldi YL, low(SUB16_OP2)  ; ($0116)
747
748      ldi oloop, 2      ; load oloop with 2 to loop 2 times.
749 comploadloop2:
750      lpm mpr, Z+ ; load mpr from Z, inc Z
751      st Y+, mpr ; store mpr to Y, inc Y
752      dec oloop      ; decrement oloop to run loop 2 times
753      brne comploadloop2
754      ; Operand H now loaded to SUB16_OP2
755
756      ldi ZH, high(OperandI) ; load OperandD location to Z
757      ldi ZL, low(OperandI)
758      ; Shift Z to prepare for program memory access:
759      lsl ZH
760      lsl ZL
761      adc ZH, zero ; shift carry from lower byte to upper byte
762      ldi YH, high(ADD16_OP2) ; Load OP2 location into Y
763      ldi YL, low(ADD16_OP2) ; ($0112)
764
765      ldi oloop, 2      ; load oloop with 2 to loop 2 times.
766 comploadloop3:
767      lpm mpr, Z+ ; load mpr from Z, inc Z
768      st Y+, mpr ; store mpr to Y, inc Y
769      dec oloop      ; decrement oloop to run loop 2 times
770      brne comploadloop3
771      ; Operand I now loaded to ADD16_OP2
772
773      pop oloop      ; pop regs from stack
774      pop iloop
775      pop mpr
776      pop ZL
777      pop ZH
778      pop YL
779      pop YH
780
781      ret                      ; End a function with RET
782
783 ;-----
784 ; Func: CLRRES
785 ; Desc: Clears the memory locations of the results for
786 ;       ADD16, SUB16, and MUL24
787 ;-----
788 CLRRES:
789      push XL
790      push XH
791      push YL

```

```

792     push YH
793     push ZL
794     push ZH
795
796     ; Load beginning address of ADD16 result to Z
797     ldi    XL, low(ADD16_Result)    ; Load low byte of address
798     ldi    XH, high(ADD16_Result)   ; Load high byte of address
799     ; Load beginning address of SUB16 result to Y
800     ldi    YL, low(SUB16_Result)    ; Load low byte of address
801     ldi    YH, high(SUB16_Result)   ; Load high byte of address
802     ; Load beginning address of MUL24 result to Z
803     ldi    ZL, low(MUL24_Result)
804     ldi    ZH, high(MUL24_Result)
805
806     ; Write zeros to all result locations
807     st X+, zero
808     st X+, zero
809     st X, zero    ; Three bytes for ADD16 result
810     st Y+, zero
811     st Y, zero    ; Two for SUB16
812     st Z+, zero
813     st Z+, zero
814     st Z+, zero
815     st Z+, zero
816     st Z+, zero
817     st Z, zero    ; And SIX for MUL24
818
819     pop ZH
820     pop ZL
821     pop YH
822     pop YL
823     pop XH
824     pop XL
825
826     ret
827
828 ;-----
829 ; Func: MUL16
830 ; Desc: An example function that multiplies two 16-bit numbers
831 ;       A – Operand A is gathered from address $0101:$0100
832 ;       B – Operand B is gathered from address $0103:$0102
833 ;       Res – Result is stored in address
834 ;           $0107:$0106:$0105:$0104
835 ;       You will need to make sure that Res is cleared before
836 ;       calling this function.
837 ;-----

```

```

838 MUL16:
839     push    A                ; Save A register
840     push    B                ; Save B register
841     push    rhi              ; Save rhi register
842     push    rlo              ; Save rlo register
843     push    zero             ; Save zero register
844     push    XH               ; Save X-ptr
845     push    XL
846     push    YH                ; Save Y-ptr
847     push    YL
848     push    ZH                ; Save Z-ptr
849     push    ZL
850     push    oloop             ; Save counters
851     push    iloop
852
853     clr     zero              ; Maintain zero semantics
854
855     ; Set Y to beginning address of B
856     ldi     YL, low(addrB)    ; Load low byte
857     ldi     YH, high(addrB)   ; Load high byte
858
859     ; Set Z to beginning address of resulting Product
860     ldi     ZL, low(LAddrP)   ; Load low byte
861     ldi     ZH, high(LAddrP)  ; Load high byte
862
863     ; Begin outer for loop
864     ldi     oloop, 2           ; Load counter
865 MUL16_OLOOP:
866     ; Set X to beginning address of A
867     ldi     XL, low(addrA)    ; Load low byte
868     ldi     XH, high(addrA)   ; Load high byte
869
870     ; Begin inner for loop
871     ldi     iloop, 2           ; Load counter
872 MUL16_ILOOP:
873     ld      A, X+              ; Get byte of A operand
874     ld      B, Y               ; Get byte of B operand
875     mul     A,B                ; Multiply A and B
876     ld      A, Z+              ; Get a result byte from memory
877     ld      B, Z+              ; Get the next result byte from memory
878     add     rlo, A              ; rlo <= rlo + A
879     adc     rhi, B              ; rhi <= rhi + B + carry
880     ld      A, Z               ; Get a third byte from the result
881     adc     A, zero             ; Add carry to A
882     st      Z, A               ; Store third byte to memory
883     st      -Z, rhi            ; Store second byte to memory

```



```

884      st      -Z, rlo          ; Store first byte to memory
885      adiw    ZH:ZL, 1         ; Z <= Z + 1
886      dec     iloop           ; Decrement counter
887      brne    MUL16_ILOOP     ; Loop if iLoop != 0
888      ; End inner for loop
889
890      sbiw    ZH:ZL, 1         ; Z <= Z - 1
891      adiw    YH:YL, 1         ; Y <= Y + 1
892      dec     oloop           ; Decrement counter
893      brne    MUL16_OLOOP     ; Loop if oLoop != 0
894      ; End outer for loop
895
896      ; Restore all registers in reverse order
897      pop     iloop
898      pop     oloop
899      pop     ZL
900      pop     ZH
901      pop     YL
902      pop     YH
903      pop     XL
904      pop     XH
905      pop     zero
906      pop     rlo
907      pop     rhi
908      pop     B
909      pop     A
910      ret                          ; End a function with RET
911
912 ;-----
913 ; Func: Template function header
914 ; Desc: Cut and paste this and fill in the info at the
915 ;       beginning of your functions
916 ;-----
917 FUNC:                          ; Begin a function with a label
918      ; Save variable by pushing them to the stack
919
920      ; Execute the function here
921
922      ; Restore variable by popping them from the stack
923      ret                          ; End a function with RET
924
925
926 ; *****
927 ; *   Stored Program Data
928 ; *   Do not section.
929 ; *****

```

```

930 ; ADD16 operands
931 OperandA:
932     .DW 0xFCBA
933 OperandB:
934     .DW 0xFFFF
935
936 ; SUB16 operands
937 OperandC:
938     .DW 0xFCB9
939 OperandD:
940     .DW 0XE420
941
942 ; MUL24 operands
943 OperandE1:
944     .DW 0xFFFF
945 OperandE2:
946     .DW 0X00FF
947 OperandF1:
948     .DW 0xFFFF
949 OperandF2:
950     .DW 0X00FF
951
952 ; Compound operands
953 OperandG:
954     .DW 0xFCBA                ; test value for operand G
955 OperandH:
956     .DW 0x2022                ; test value for operand H
957 OperandI:
958     .DW 0x21BB                ; test value for operand I
959
960 ; *****
961 ; *   Data Memory Allocation
962 ; *****
963 .dseg
964 .org    $0100                ; data memory allocation for MUL16 example
965 addrA:  .byte 2
966 addrB:  .byte 2
967 LAddrP: .byte 4
968
969 ; Below is an example of data memory allocation for ADD16.
970 ; Consider using something similar for SUB16 and MUL24.
971 .org    $0110                ; data memory allocation for operands
972 ADD16_OP1:  ; $0110
973     .byte 2                ; allocate two bytes for first operand of ADD16
974 ADD16_OP2:  ; $0112
975     .byte 2                ; allocate two bytes for second operand of ADD16

```

```

976 SUB16_OP1:  ;$0114
977     .byte 2      ; allocate two bytes for first operand of SUB16
978 SUB16_OP2:  ;$0116
979     .byte 2      ; allocate two bytes for second operand of SUB16
980 MUL24_OP1:  ;$0118
981     .byte 3      ; allocate three bytes for first operand of MUL24
982 MUL24_OP2:  ;$011B
983     .byte 3      ; allocate three bytes for second operand of MUL24
984
985
986 .org      $0120      ; data memory allocation for results
987 ADD16_Result:
988     .byte 3      ; allocate three bytes for ADD16 result
989 .org      $0130
990 SUB16_Result:
991     .byte 2      ; allocate two bytes for SUB16 result
992 .org      $0140
993 MUL24_Result:
994     .byte 6      ; allocate six bytes for MUL24 result
995
996 ;*****
997 ;*   Additional Program Includes
998 ;*****
999 ; There are no additional file includes for this program

```