

c. What are the states of the V, N, and C flags after the second adds instruction?

```
1 0 Assume you are using signed 32 bit signed numbers
2 0 Deliverable1:What are the states of the V, N, and C flags after the first adds instruction?
3 0 Deliverable2:For the first adds, what is the expected and actual result? Is the result correct?
4 0 Deliverable3:What are the states of the V, N, and C flags after the second adds instruction?
5 0 Deliverable4:For the second adds, what is the expected and actual result? Is the result correct?
6 0 Deliverable5:What are the states of the V, N, and C flags after the third adds instruction?
7 0 Deliverable6:For the third adds, what is the expected and actual result? Is the result correct?
8 0 Deliverable7:What are the states of the V, N, and C flags after the fourth adds instruction?
9 0 Deliverable8:For the fourth adds, what is the expected and actual result? Is the result correct?
10 0 Deliverable9:What do the V, N, and C flags tell you?
11
12 .data
13 data1: .word 0x0E2F356F
14 data2: .word 0x13D49530
15 data3: .word 0x542F356F
16
17 cpsr
18 fpcsr
19 (gdb) n
20 str r4, [r3] @store result in location
21 (gdb) n
22 ldr r1, =data3
23 (gdb) n
24 ldr r2, =data4
25 (gdb) n
26 ldr r3, =result2
27 (gdb) n
28 ldr r4, [r1] @ r4 = ??
29 (gdb) n
30 ldr r5, [r2] @ r5 = ??
31 (gdb) n
32 adds r4, r4, r5 @ r4 = r4 + r5 = ???, or does it?
33 (gdb) i r
34 r0 0x0 0
35 r1 0x20124 131364
36 r2 0x20128 131368
37 r3 0x20140 131392
38 r4 0x542F356F 1412380015
39 r5 0x1269530 116708144
40 r6 0x0 0
41 r7 0x0 0
42 r8 0x0 0
43 r9 0x0 0
44 r10 0x0 0
45 r11 0x0 0
46 r12 0x0 0
47 sp 0xffffef0 0xffffef0
48 lr 0x0 0
49 pc 0x10004 <_start+48>
50 cpsr 0x90000010 -1879048176
51 fpcsr 0x0 0
52 (gdb) 
```

d. For the second adds, what is the expected and actual result? Is the result correct?

The expected result is 262732, but the given result is 1412380015. This is not correct.

e. What are the states of the V, N, and C flags after the third adds instruction?

```
1 0 Assume you are using signed 32 bit signed numbers
2 0 Deliverable1:What are the states of the V, N, and C flags after the first adds instruction?
3 0 Deliverable2:For the first adds, what is the expected and actual result? Is the result correct?
4 0 Deliverable3:What are the states of the V, N, and C flags after the second adds instruction?
5 0 Deliverable4:For the second adds, what is the expected and actual result? Is the result correct?
6 0 Deliverable5:What are the states of the V, N, and C flags after the third adds instruction?
7 0 Deliverable6:For the third adds, what is the expected and actual result? Is the result correct?
8 0 Deliverable7:What are the states of the V, N, and C flags after the fourth adds instruction?
9 0 Deliverable8:For the fourth adds, what is the expected and actual result? Is the result correct?
10 0 Deliverable9:What do the V, N, and C flags tell you?
11
12 .data
13 data1: .word 0x0E2F356F
14 data2: .word 0x13D49530
15 data3: .word 0x542F356F
16
17 cpsr
18 fpcsr
19 (gdb) n
20 str r4, [r3] @store result in location
21 (gdb) n
22 ldr r1, =data5
23 (gdb) n
24 ldr r2, =data6
25 (gdb) n
26 ldr r3, =result3
27 (gdb) n
28 ldr r4, [r1] @ r4 = ??
29 (gdb) n
30 ldr r5, [r2] @ r5 = ??
31 (gdb) n
32 adds r4, r4, r5 @ r4 = r4 + r5 = ???, or does it?
33 (gdb) i r
34 r0 0x0 0
35 r1 0x2012c 131372
36 r2 0x20130 131376
37 r3 0x20144 131396
38 r4 0x90000000 2147483648
39 r5 0xffffffff 4294967295
40 r6 0x0 0
41 r7 0x0 0
42 r8 0x0 0
43 r9 0x0 0
44 r10 0x0 0
45 r11 0x0 0
46 r12 0x0 0
47 sp 0xffffef0 0xffffef0
48 lr 0x0 0
49 pc 0x1000c <_start+78>
50 cpsr 0x10 16
51 fpcsr 0x0 0
52 (gdb) 
```

f. For the third adds, what is the expected and actual result? Is the result correct?

The expected result is 262,748, but is instead the much higher 2147483648.

g. What are the states of the V, N, and C flags after the fourth adds instruction?

The screenshot shows a Visual Studio Code window with a C program named `1.flags_ex5_13.s`. The program contains several deliverable questions and assembly code. The assembly code includes instructions for loading data, adding registers, and checking flags. The terminal output shows the execution of the program, with the final result being 2147483648, which is incorrect for the given inputs.

```
1 // Assume you are using signed 32 bit signed numbers
2 // Deliverable1: What are the states of the V, N, and C flags after the first adds instruction?
3 // Deliverable2: For the first adds, what is the expected and actual result? Is the result correct?
4 // Deliverable3: What are the states of the V, N, and C flags after the second adds instruction?
5 // Deliverable4: For the second adds, what is the expected and actual result? Is the result correct?
6 // Deliverable5: What are the states of the V, N, and C flags after the third adds instruction?
7 // Deliverable6: For the third adds, what is the expected and actual result? Is the result correct?
8 // Deliverable7: What are the states of the V, N, and C flags after the fourth adds instruction?
9 // Deliverable8: For the fourth adds, what is the expected and actual result? Is the result correct?
10 // Deliverable9: What do the V, N, and C flags tell you?
11
12 .data
13 data1: .word 0x02F35F
14 data2: .word 0x13D49530
15 data3: .word 0x542F35F
16
17 .text
18 @Store result in location
19 str r4, [r3]
20
21 ldr r1, =data7
22 ldr r2, =data8
23 ldr r3, =result4
24 ldr r4, [r1] @ r4 = ??
25 ldr r5, [r2] @ r5 = ??
26 adds r4, r4, r5 @ r4 = r4 + r5 = ???, or does it?
27
28 @ Print the result
29 mov r0, #0
30 mov r1, #131380
31 mov r2, #131384
32 mov r3, #131400
33 mov r4, #4294967294
34 mov r5, #4294967291
35
36 mov r6, #0
37 mov r7, #0
38 mov r8, #0
39 mov r9, #0
40 mov r10, #0
41 mov r11, #0
42 mov r12, #0
43
44 sp, 0xffffef00
45 lr, 0
46 pc, 0x100dc_start+194>
47 cpsr, 0x30000010
48 fpcsr, 0
49 (gdb) i r
50 r0 0x0 0
51 r1 0x29134 131380
52 r2 0x29138 131384
53 r3 0x29140 131400
54 r4 0xffffffff 4294967294
55 r5 0xffffffff 4294967291
56 r6 0x0 0
57 r7 0x0 0
58 r8 0x0 0
59 r9 0x0 0
60 r10 0x0 0
61 r11 0x0 0
62 r12 0x0 0
63 sp 0xffffef00 0xffffef00
64 lr 0x0 0
65 pc 0x100dc_start+194> 805306384
66 cpsr 0x30000010 0
67 fpcsr 0
68 (gdb) i r
69 r0 0x0 0
70 r1 0x29134 131380
71 r2 0x29138 131384
72 r3 0x29140 131400
73 r4 0xffffffff 4294967294
74 r5 0xffffffff 4294967291
75 r6 0x0 0
76 r7 0x0 0
77 r8 0x0 0
78 r9 0x0 0
79 r10 0x0 0
80 r11 0x0 0
81 r12 0x0 0
82 sp 0xffffef00 0xffffef00
83 lr 0x0 0
84 pc 0x100dc_start+194> 805306384
85 cpsr 0x30000010 0
86 fpcsr 0
87 (gdb) i r
88 r0 0x0 0
89 r1 0x29134 131380
90 r2 0x29138 131384
91 r3 0x29140 131400
92 r4 0xffffffff 4294967294
93 r5 0xffffffff 4294967291
94 r6 0x0 0
95 r7 0x0 0
96 r8 0x0 0
97 r9 0x0 0
98 r10 0x0 0
99 r11 0x0 0
100 r12 0x0 0
101 sp 0xffffef00 0xffffef00
102 lr 0x0 0
103 pc 0x100dc_start+194> 805306384
104 cpsr 0x30000010 0
105 fpcsr 0
106 (gdb) i r
107 r0 0x0 0
108 r1 0x29134 131380
109 r2 0x29138 131384
110 r3 0x29140 131400
111 r4 0xffffffff 4294967294
112 r5 0xffffffff 4294967291
113 r6 0x0 0
114 r7 0x0 0
115 r8 0x0 0
116 r9 0x0 0
117 r10 0x0 0
118 r11 0x0 0
119 r12 0x0 0
120 sp 0xffffef00 0xffffef00
121 lr 0x0 0
122 pc 0x100dc_start+194> 805306384
123 cpsr 0x30000010 0
124 fpcsr 0
125 (gdb) i r
126 r0 0x0 0
127 r1 0x29134 131380
128 r2 0x29138 131384
129 r3 0x29140 131400
130 r4 0xffffffff 4294967294
131 r5 0xffffffff 4294967291
132 r6 0x0 0
133 r7 0x0 0
134 r8 0x0 0
135 r9 0x0 0
136 r10 0x0 0
137 r11 0x0 0
138 r12 0x0 0
139 sp 0xffffef00 0xffffef00
140 lr 0x0 0
141 pc 0x100dc_start+194> 805306384
142 cpsr 0x30000010 0
143 fpcsr 0
144 (gdb) i r
145 r0 0x0 0
146 r1 0x29134 131380
147 r2 0x29138 131384
148 r3 0x29140 131400
149 r4 0xffffffff 4294967294
150 r5 0xffffffff 4294967291
151 r6 0x0 0
152 r7 0x0 0
153 r8 0x0 0
154 r9 0x0 0
155 r10 0x0 0
156 r11 0x0 0
157 r12 0x0 0
158 sp 0xffffef00 0xffffef00
159 lr 0x0 0
160 pc 0x100dc_start+194> 805306384
161 cpsr 0x30000010 0
162 fpcsr 0
163 (gdb) i r
164 r0 0x0 0
165 r1 0x29134 131380
166 r2 0x29138 131384
167 r3 0x29140 131400
168 r4 0xffffffff 4294967294
169 r5 0xffffffff 4294967291
170 r6 0x0 0
171 r7 0x0 0
172 r8 0x0 0
173 r9 0x0 0
174 r10 0x0 0
175 r11 0x0 0
176 r12 0x0 0
177 sp 0xffffef00 0xffffef00
178 lr 0x0 0
179 pc 0x100dc_start+194> 805306384
180 cpsr 0x30000010 0
181 fpcsr 0
182 (gdb) i r
183 r0 0x0 0
184 r1 0x29134 131380
185 r2 0x29138 131384
186 r3 0x29140 131400
187 r4 0xffffffff 4294967294
188 r5 0xffffffff 4294967291
189 r6 0x0 0
190 r7 0x0 0
191 r8 0x0 0
192 r9 0x0 0
193 r10 0x0 0
194 r11 0x0 0
195 r12 0x0 0
196 sp 0xffffef00 0xffffef00
197 lr 0x0 0
198 pc 0x100dc_start+194> 805306384
199 cpsr 0x30000010 0
200 fpcsr 0
201 (gdb) i r
202 r0 0x0 0
203 r1 0x29134 131380
204 r2 0x29138 131384
205 r3 0x29140 131400
206 r4 0xffffffff 4294967294
207 r5 0xffffffff 4294967291
208 r6 0x0 0
209 r7 0x0 0
210 r8 0x0 0
211 r9 0x0 0
212 r10 0x0 0
213 r11 0x0 0
214 r12 0x0 0
215 sp 0xffffef00 0xffffef00
216 lr 0x0 0
217 pc 0x100dc_start+194> 805306384
218 cpsr 0x30000010 0
219 fpcsr 0
220 (gdb) i r
221 r0 0x0 0
222 r1 0x29134 131380
223 r2 0x29138 131384
224 r3 0x29140 131400
225 r4 0xffffffff 4294967294
226 r5 0xffffffff 4294967291
227 r6 0x0 0
228 r7 0x0 0
229 r8 0x0 0
230 r9 0x0 0
231 r10 0x0 0
232 r11 0x0 0
233 r12 0x0 0
234 sp 0xffffef00 0xffffef00
235 lr 0x0 0
236 pc 0x100dc_start+194> 805306384
237 cpsr 0x30000010 0
238 fpcsr 0
239 (gdb) i r
240 r0 0x0 0
241 r1 0x29134 131380
242 r2 0x29138 131384
243 r3 0x29140 131400
244 r4 0xffffffff 4294967294
245 r5 0xffffffff 4294967291
246 r6 0x0 0
247 r7 0x0 0
248 r8 0x0 0
249 r9 0x0 0
250 r10 0x0 0
251 r11 0x0 0
252 r12 0x0 0
253 sp 0xffffef00 0xffffef00
254 lr 0x0 0
255 pc 0x100dc_start+194> 805306384
256 cpsr 0x30000010 0
257 fpcsr 0
258 (gdb) i r
259 r0 0x0 0
260 r1 0x29134 131380
261 r2 0x29138 131384
262 r3 0x29140 131400
263 r4 0xffffffff 4294967294
264 r5 0xffffffff 4294967291
265 r6 0x0 0
266 r7 0x0 0
267 r8 0x0 0
268 r9 0x0 0
269 r10 0x0 0
270 r11 0x0 0
271 r12 0x0 0
272 sp 0xffffef00 0xffffef00
273 lr 0x0 0
274 pc 0x100dc_start+194> 805306384
275 cpsr 0x30000010 0
276 fpcsr 0
277 (gdb) i r
278 r0 0x0 0
279 r1 0x29134 131380
280 r2 0x29138 131384
281 r3 0x29140 131400
282 r4 0xffffffff 4294967294
283 r5 0xffffffff 4294967291
284 r6 0x0 0
285 r7 0x0 0
286 r8 0x0 0
287 r9 0x0 0
288 r10 0x0 0
289 r11 0x0 0
290 r12 0x0 0
291 sp 0xffffef00 0xffffef00
292 lr 0x0 0
293 pc 0x100dc_start+194> 805306384
294 cpsr 0x30000010 0
295 fpcsr 0
296 (gdb) i r
297 r0 0x0 0
298 r1 0x29134 131380
299 r2 0x29138 131384
300 r3 0x29140 131400
301 r4 0xffffffff 4294967294
302 r5 0xffffffff 4294967291
303 r6 0x0 0
304 r7 0x0 0
305 r8 0x0 0
306 r9 0x0 0
307 r10 0x0 0
308 r11 0x0 0
309 r12 0x0 0
310 sp 0xffffef00 0xffffef00
311 lr 0x0 0
312 pc 0x100dc_start+194> 805306384
313 cpsr 0x30000010 0
314 fpcsr 0
315 (gdb) i r
316 r0 0x0 0
317 r1 0x29134 131380
318 r2 0x29138 131384
319 r3 0x29140 131400
320 r4 0xffffffff 4294967294
321 r5 0xffffffff 4294967291
322 r6 0x0 0
323 r7 0x0 0
324 r8 0x0 0
325 r9 0x0 0
326 r10 0x0 0
327 r11 0x0 0
328 r12 0x0 0
329 sp 0xffffef00 0xffffef00
330 lr 0x0 0
331 pc 0x100dc_start+194> 805306384
332 cpsr 0x30000010 0
333 fpcsr 0
334 (gdb) i r
335 r0 0x0 0
336 r1 0x29134 131380
337 r2 0x29138 131384
338 r3 0x29140 131400
339 r4 0xffffffff 4294967294
340 r5 0xffffffff 4294967291
341 r6 0x0 0
342 r7 0x0 0
343 r8 0x0 0
344 r9 0x0 0
345 r10 0x0 0
346 r11 0x0 0
347 r12 0x0 0
348 sp 0xffffef00 0xffffef00
349 lr 0x0 0
350 pc 0x100dc_start+194> 805306384
351 cpsr 0x30000010 0
352 fpcsr 0
353 (gdb) i r
354 r0 0x0 0
355 r1 0x29134 131380
356 r2 0x29138 131384
357 r3 0x29140 131400
358 r4 0xffffffff 4294967294
359 r5 0xffffffff 4294967291
360 r6 0x0 0
361 r7 0x0 0
362 r8 0x0 0
363 r9 0x0 0
364 r10 0x0 0
365 r11 0x0 0
366 r12 0x0 0
367 sp 0xffffef00 0xffffef00
368 lr 0x0 0
369 pc 0x100dc_start+194> 805306384
370 cpsr 0x30000010 0
371 fpcsr 0
372 (gdb) i r
373 r0 0x0 0
374 r1 0x29134 131380
375 r2 0x29138 131384
376 r3 0x29140 131400
377 r4 0xffffffff 4294967294
378 r5 0xffffffff 4294967291
379 r6 0x0 0
380 r7 0x0 0
381 r8 0x0 0
382 r9 0x0 0
383 r10 0x0 0
384 r11 0x0 0
385 r12 0x0 0
386 sp 0xffffef00 0xffffef00
387 lr 0x0 0
388 pc 0x100dc_start+194> 805306384
389 cpsr 0x30000010 0
390 fpcsr 0
391 (gdb) i r
392 r0 0x0 0
393 r1 0x29134 131380
394 r2 0x29138 131384
395 r3 0x29140 131400
396 r4 0xffffffff 4294967294
397 r5 0xffffffff 4294967291
398 r6 0x0 0
399 r7 0x0 0
400 r8 0x0 0
401 r9 0x0 0
402 r10 0x0 0
403 r11 0x0 0
404 r12 0x0 0
405 sp 0xffffef00 0xffffef00
406 lr 0x0 0
407 pc 0x100dc_start+194> 805306384
408 cpsr 0x30000010 0
409 fpcsr 0
410 (gdb) i r
411 r0 0x0 0
412 r1 0x29134 131380
413 r2 0x29138 131384
414 r3 0x29140 131400
415 r4 0xffffffff 4294967294
416 r5 0xffffffff 4294967291
417 r6 0x0 0
418 r7 0x0 0
419 r8 0x0 0
420 r9 0x0 0
421 r10 0x0 0
422 r11 0x0 0
423 r12 0x0 0
424 sp 0xffffef00 0xffffef00
425 lr 0x0 0
426 pc 0x100dc_start+194> 805306384
427 cpsr 0x30000010 0
428 fpcsr 0
429 (gdb) i r
430 r0 0x0 0
431 r1 0x29134 131380
432 r2 0x29138 131384
433 r3 0x29140 131400
434 r4 0xffffffff 4294967294
435 r5 0xffffffff 4294967291
436 r6 0x0 0
437 r7 0x0 0
438 r8 0x0 0
439 r9 0x0 0
440 r10 0x0 0
441 r11 0x0 0
442 r12 0x0 0
443 sp 0xffffef00 0xffffef00
444 lr 0x0 0
445 pc 0x100dc_start+194> 805306384
446 cpsr 0x30000010 0
447 fpcsr 0
448 (gdb) i r
449 r0 0x0 0
450 r1 0x29134 131380
451 r2 0x29138 131384
452 r3 0x29140 131400
453 r4 0xffffffff 4294967294
454 r5 0xffffffff 4294967291
455 r6 0x0 0
456 r7 0x0 0
457 r8 0x0 0
458 r9 0x0 0
459 r10 0x0 0
460 r11 0x0 0
461 r12 0x0 0
462 sp 0xffffef00 0xffffef00
463 lr 0x0 0
464 pc 0x100dc_start+194> 805306384
465 cpsr 0x30000010 0
466 fpcsr 0
467 (gdb) i r
468 r0 0x0 0
469 r1 0x29134 131380
470 r2 0x29138 131384
471 r3 0x29140 131400
472 r4 0xffffffff 4294967294
473 r5 0xffffffff 4294967291
474 r6 0x0 0
475 r7 0x0 0
476 r8 0x0 0
477 r9 0x0 0
478 r10 0x0 0
479 r11 0x0 0
480 r12 0x0 0
481 sp 0xffffef00 0xffffef00
482 lr 0x0 0
483 pc 0x100dc_start+194> 805306384
484 cpsr 0x30000010 0
485 fpcsr 0
486 (gdb) i r
487 r0 0x0 0
488 r1 0x29134 131380
489 r2 0x29138 131384
490 r3 0x29140 131400
491 r4 0xffffffff 4294967294
492 r5 0xffffffff 4294967291
493 r6 0x0 0
494 r7 0x0 0
495 r8 0x0 0
496 r9 0x0 0
497 r10 0x0 0
498 r11 0x0 0
499 r12 0x0 0
500 sp 0xffffef00 0xffffef00
501 lr 0x0 0
502 pc 0x100dc_start+194> 805306384
503 cpsr 0x30000010 0
504 fpcsr 0
505 (gdb) i r
506 r0 0x0 0
507 r1 0x29134 131380
508 r2 0x29138 131384
509 r3 0x29140 131400
510 r4 0xffffffff 4294967294
511 r5 0xffffffff 4294967291
512 r6 0x0 0
513 r7 0x0 0
514 r8 0x0 0
515 r9 0x0 0
516 r10 0x0 0
517 r11 0x0 0
518 r12 0x0 0
519 sp 0xffffef00 0xffffef00
520 lr 0x0 0
521 pc 0x100dc_start+194> 805306384
522 cpsr 0x30000010 0
523 fpcsr 0
524 (gdb) i r
525 r0 0x0 0
526 r1 0x29134 131380
527 r2 0x29138 131384
528 r3 0x29140 131400
529 r4 0xffffffff 4294967294
530 r5 0xffffffff 4294967291
531 r6 0x0 0
532 r7 0x0 0
533 r8 0x0 0
534 r9 0x0 0
535 r10 0x0 0
536 r11 0x0 0
537 r12 0x0 0
538 sp 0xffffef00 0xffffef00
539 lr 0x0 0
540 pc 0x100dc_start+194> 805306384
541 cpsr 0x30000010 0
542 fpcsr 0
543 (gdb) i r
544 r0 0x0 0
545 r1 0x29134 131380
546 r2 0x29138 131384
547 r3 0x29140 131400
548 r4 0xffffffff 4294967294
549 r5 0xffffffff 4294967291
550 r6 0x0 0
551 r7 0x0 0
552 r8 0x0 0
553 r9 0x0 0
554 r10 0x0 0
555 r11 0x0 0
556 r12 0x0 0
557 sp 0xffffef00 0xffffef00
558 lr 0x0 0
559 pc 0x100dc_start+194> 805306384
560 cpsr 0x30000010 0
561 fpcsr 0
562 (gdb) i r
563 r0 0x0 0
564 r1 0x29134 131380
565 r2 0x29138 131384
566 r3 0x29140 131400
567 r4 0xffffffff 4294967294
568 r5 0xffffffff 4294967291
569 r6 0x0 0
570 r7 0x0 0
571 r8 0x0 0
572 r9 0x0 0
573 r10 0x0 0
574 r11 0x0 0
575 r12 0x0 0
576 sp 0xffffef00 0xffffef00
577 lr 0x0 0
578 pc 0x100dc_start+194> 805306384
579 cpsr 0x30000010 0
580 fpcsr 0
581 (gdb) i r
582 r0 0x0 0
583 r1 0x29134 131380
584 r2 0x29138 131384
585 r3 0x29140 131400
586 r4 0xffffffff 4294967294
587 r5 0xffffffff 4294967291
588 r6 0x0 0
589 r7 0x0 0
590 r8 0x0 0
591 r9 0x0 0
592 r10 0x0 0
593 r11 0x0 0
594 r12 0x0 0
595 sp 0xffffef00 0xffffef00
596 lr 0x0 0
597 pc 0x100dc_start+194> 805306384
598 cpsr 0x30000010 0
599 fpcsr 0
600 (gdb) i r
601 r0 0x0 0
602 r1 0x29134 131380
603 r2 0x29138 131384
604 r3 0x29140 131400
605 r4 0xffffffff 4294967294
606 r5 0xffffffff 4294967291
607 r6 0x0 0
608 r7 0x0 0
609 r8 0x0 0
610 r9 0x0 0
611 r10 0x0 0
612 r11 0x0 0
613 r12 0x0 0
614 sp 0xffffef00 0xffffef00
615 lr 0x0 0
616 pc 0x100dc_start+194> 805306384
617 cpsr 0x30000010 0
618 fpcsr 0
619 (gdb) i r
620 r0 0x0 0
621 r1 0x29134 131380
622 r2 0x29138 131384
623 r3 0x29140 131400
624 r4 0xffffffff 4294967294
625 r5 0xffffffff 4294967291
626 r6 0x0 0
627 r7 0x0 0
628 r8 0x0 0
629 r9 0x0 0
630 r10 0x0 0
631 r11 0x0 0
632 r12 0x0 0
633 sp 0xffffef00 0xffffef00
634 lr 0x0 0
635 pc 0x100dc_start+194> 805306384
636 cpsr 0x30000010 0
637 fpcsr 0
638 (gdb) i r
639 r0 0x0 0
640 r1 0x29134 131380
641 r2 0x29138 131384
642 r3 0x29140 131400
643 r4 0xffffffff 4294967294
644 r5 0xffffffff 4294967291
645 r6 0x0 0
646 r7 0x0 0
647 r8 0x0 0
648 r9 0x0 0
649 r10 0x0 0
650 r11 0x0 0
651 r12 0x0 0
652 sp 0xffffef00 0xffffef00
653 lr 0x0 0
654 pc 0x100dc_start+194> 805306384
655 cpsr 0x30000010 0
656 fpcsr 0
657 (gdb) i r
658 r0 0x0 0
659 r1 0x29134 131380
660 r2 0x29138 131384
661 r3 0x29140 131400
662 r4 0xffffffff 4294967294
663 r5 0xffffffff 4294967291
664 r6 0x0 0
665 r7 0x0 0
666 r8 0x0 0
667 r9 0x0 0
668 r10 0x0 0
669 r11 0x0 0
670 r12 0x0 0
671 sp 0xffffef00 0xffffef00
672 lr 0x0 0
673 pc 0x100dc_start+194> 805306384
674 cpsr 0x30000010 0
675 fpcsr 0
676 (gdb) i r
677 r0 0x0 0
678 r1 0x29134 131380
679 r2 0x29138 131384
680 r3 0x29140 131400
681 r4 0xffffffff 4294967294
682 r5 0xffffffff 4294967291
683 r6 0x0 0
684 r7 0x0 0
685 r8 0x0 0
686 r9 0x0 0
687 r10 0x0 0
688 r11 0x0 0
689 r12 0x0 0
690 sp 0xffffef00 0xffffef00
691 lr 0x0 0
692 pc 0x100dc_start+194> 805306384
693 cpsr 0x30000010 0
694 fpcsr 0
695 (gdb) i r
696 r0 0x0 0
697 r1 0x29134 131380
698 r2 0x29138 131384
699 r3 0x29140 131400
700 r4 0xffffffff 4294967294
701 r5 0xffffffff 4294967291
702 r6 0x0 0
703 r7 0x0 0
704 r8 0x0 0
705 r9 0x0 0
706 r10 0x0 0
707 r11 0x0 0
708 r12 0x0 0
709 sp 0xffffef00 0xffffef00
710 lr 0x0 0
711 pc 0x100dc_start+194> 805306384
712 cpsr 0x30000010 0
713 fpcsr 0
714 (gdb) i r
715 r0 0x0 0
716 r1 0x29134 131380
717 r2 0x29138 131384
718 r3 0x29140 131400
719 r4 0xffffffff 4294967294
720 r5 0xffffffff 4294967291
721 r6 0x0 0
722 r7 0x0 0
723 r8 0x0 0
724 r9 0x0 0
725 r10 0x0 0
726 r11 0x0 0
727 r12 0x0 0
728 sp 0xffffef00 0xffffef00
729 lr 0x0 0
730 pc 0x100dc_start+194> 805306384
731 cpsr 0x30000010 0
732 fpcsr 0
733 (gdb) i r
734 r0 0x0 0
735 r1 0x29134 131380
736 r2 0x29138 131384
737 r3 0x29140 131400
738 r4 0xffffffff 4294967294
739 r5 0xffffffff 4294967291
740 r6 0x0 0
741 r7 0x0 0
742 r8 0x0 0
743 r9 0x0 0
744 r10 0x0 0
745 r11 0x0 0
746 r12 0x0 0
747 sp 0xffffef00 0xffffef00
748 lr 0x0 0
749 pc 0x100dc_start+194> 805306384
750 cpsr 0x30000010 0
751 fpcsr 0
752 (gdb) i r
753 r0 0x0 0
754 r1 0x29134 131380
755 r2 0x29138 131384
756 r3 0x29140 131400
757 r4 0xffffffff 4294967294
758 r5 0xffffffff 4294967291
759 r6 0x0 0
760 r7 0x0 0
761 r8 0x0 0
762 r9 0x0 0
763 r10 0x0 0
764 r11 0x0 0
765 r12 0x0 0
766 sp 0xffffef00 0xffffef00
767 lr 0x0 0
768 pc 0x100dc_start+194> 805306384
769 cpsr 0x30000010 0
770 fpcsr 0
771 (gdb) i r
772 r0 0x0 0
773 r1 0x29134 131380
774 r2 0x29138 131384
775 r3 0x29140 131400
776 r4 0xffffffff 4294967294
777 r5 0xffffffff 4294967291
778 r6 0x0 0
779 r7 0x0 0
780 r8 0x0 0
781 r9 0x0 0
782 r10 0x0 
```

```

1 0 This program looks at 8-bit signed numbers and how to use the sign extended instructions. Make sure you consider the answers as 8-bit.
2 0 Deliverable 1: What are the V, N, and C flags after the first adds instruction executes?
3 0 Deliverable 2: What is the expected and actual result? Is it correct?
4 0 Deliverable 3: Now, look at the second example. What does the ldsrb instruction do?
5 0 Deliverable 4: What are the V, N, and C flags after the second adds instruction?
6 0 Deliverable 5: What is the expected and actual result of the second adds? Is it correct?
7
8
9 .data
10 data1: .byte -128
11 data2: .byte -2
12 result: .byte 0
13
14 .text
15 .global _start
16
17
18
19
20
21 Breakpoint 1 at 0x10074: file 2_flagsfixed_ex5_13.s, line 21.
(gdb) run
Starting program: /home/group5/Desktop/Group-5/Edvin_Assembly/Signed/2_flagsfixed_ex5_13
21
(gdb) n
22 ldr r1, =data1
23 ldr r2, =data2
24 ldr r3, =result
25 ldrb r4, [r1] @ r4 = -128
26 ldrb r5, [r2] @ r5 = -2
27 adds r4, r4, r5 @ r4 = r4 + r5 = ???
28
(gdb) i r
r0 0x0 0
r1 0x20004 131252
r2 0x20005 131253
r3 0x20006 131254
r4 0x80 128
r5 0xfe 254
r6 0x0 0
r7 0x0 0
r8 0x0 0
r9 0x0 0
r10 0x0 0
r11 0x0 0
r12 0x0 0
sp 0xfffff000 0xfffff000
lr 0x0 0
pc 0x10008 0x10008 <_start+20>
cvar 0x10 16
fpvar 0x0 0
(gdb)

```

b. What is the expected and actual result? Is it correct?

The expected result is -130, but is given as 128.

c. Now, look at the second example. What does the ldsrb instruction do?

The LDSRB loads a double-word instruction into the registers.

```

1 0 This program looks at 8-bit signed numbers and how to use the sign extended instructions. Make sure you consider the answers as 8-bit.
2 0 Deliverable 1: What are the V, N, and C flags after the first adds instruction executes?
3 0 Deliverable 2: What is the expected and actual result? Is it correct?
4 0 Deliverable 3: Now, look at the second example. What does the ldsrb instruction do?
5 0 Deliverable 4: What are the V, N, and C flags after the second adds instruction?
6 0 Deliverable 5: What is the expected and actual result of the second adds? Is it correct?
7
8
9 .data
10 data1: .byte -128
11 data2: .byte -2
12 result: .byte 0
13
14 .text
15 .global _start
16
17
18
19
20
21 Breakpoint 1 at 0x10074: file 2_flagsfixed_ex5_13.s, line 21.
(gdb) run
Starting program: /home/group5/Desktop/Group-5/Edvin_Assembly/Signed/2_flagsfixed_ex5_13
21
(gdb) n
22 ldr r1, =data1
23 ldr r2, =data2
24 ldr r3, =result
25 ldrb r4, [r1] @ r4 = -128
26 ldrb r5, [r2] @ r5 = -2
27 adds r4, r4, r5 @ r4 = r4 + r5 = ???
28
(gdb) i r
r0 0x0 0
r1 0x20004 131252
r2 0x20005 131253
r3 0x20006 131254
r4 0xfffff000 4294967168
r5 0xfffff000 4294967168
r6 0x0 0
r7 0x0 0
r8 0x0 0
r9 0x0 0
r10 0x0 0
r11 0x0 0
r12 0x0 0
sp 0xfffff000 0xfffff000
lr 0x0 0
pc 0x10008 0x10008 <_start+20>
cvar 0x10 16
fpvar 0x0 0
(gdb) n
29 str r4, [r3] @store result in location result
30
31
32 ldrsb r4, [r1] @ r4 = -128
33 ldrsb r5, [r2] @ r5 = -2
34 adds r4, r4, r5 @ r4 = r4 + r5 = ???
35
(gdb) i r
r0 0x0 0
r1 0x20004 131252
r2 0x20005 131253
r3 0x20006 131254
r4 0xfffff000 4294967168
r5 0xfffff000 4294967168
r6 0x0 0
r7 0x0 0
r8 0x0 0
r9 0x0 0
r10 0x0 0
r11 0x0 0
r12 0x0 0
sp 0xfffff000 0xfffff000
lr 0x0 0
pc 0x10008 0x10008 <_start+36>
cvar 0x10 16
fpvar 0x0 0
(gdb)

```

d. What are the V, N, and C flags after the second adds instruction?

```

1 0 This program looks at 8-bit signed numbers and how to use the sign extended instructions. Make sure you consider the answers as 8-bit.
2 0 Deliverable 1: What are the V, N, and C flags after the first adds instruction executes?
3 0 Deliverable 2: What is the expected and actual result? Is it correct?
4 0 Deliverable 3: Now, look at the second example. What does the ldrsb instruction do?
5 0 Deliverable 4: What are the V, N, and C flags after the second adds instruction?
6 0 Deliverable 5: What is the expected and actual result of the second adds? Is it correct?
7
8
9 .data
10 data1: .byte -128
11 data2: .byte -2
12 result: .byte 0
13
14 .text
15 .global _start
16
(gdb) n
27      adds    r4, r4, r5    @ r4 = r4 + r5 = ???
(gdb) n
28      str     r4, [r3]      @store result in location result
(gdb) n
29      ldrsb   r4, [r1]      @ r4 = -128
(gdb) n
30      ldrsb   r5, [r2]      @ r5 = -2
(gdb) n
31      adds    r4, r4, r5    @ r4 = r4 + r5 = ???
(gdb) n
32      str     r4, [r3]      @store result in location
(gdb) n
33      mov     r7, #1
(gdb) n
34      svc     0
(gdb) i r
r0      0x0          0
r1      0x20004     131282
r2      0x20005     131283
r3      0x20006     131284
r4      0xffffffff  4294967295
r5      0xffffffff  4294967295
r6      0x0          0
r7      0x1          1
r8      0x0          0
r9      0x0          0
r10     0x0          0
r11     0x0          0
r12     0x0          0
lr      0xffffffff  0
pc      0x10004     0x10004 <_start+48>
cpsr    0x00000010 -1610612720
fpisr   0x0          0
(gdb)

```

e. What is the expected and actual result of the second adds? Is it correct?

The actual result is 4294967166, instead of the expected 262505

3. Loop Add

a. What is the expected result of the sum of the numbers?

b. Describe how the program works.

The program sums the signed values in the array by adding each value to an accumulator.

c. What is the bne instruction?

The BNE is a call-back instruction which calls back to a previous branch. This makes it useful for creating loops.

d. What is the value of the Z flag when the program loops? and when it does not loop?

The z-flag is 0 when it loops and stops looping when the counter at register 3 reaches zero and trips the z-flag.

```

1  @ This program will sum together a group of signed numbers.
2  @ This program also introduces our first loop using the bne instruction.
3  @ Deliverable 1: What is the expected result of the sum of the numbers?
4  @ Deliverable 2: Describe how the program works
5  @ Deliverable 3: What is the bne instruction?
6  @ Deliverable 4: What is the value of the Z flag when the program loops? and when it does not loop?
7
8
9  .data
10 sign.dat: .byte    +13, -10, +19, +14, -18, -9, +12, -19, +10
11 sum:      .word    0
12
13 .text
14
15 .global _start
16
17
18
19
20
21
22
23
24 (gdb) n
25      add    r0, r0, #1    @ point to next
26 (gdb) n
27      subs   r3, r3, #1    @ decrement counter
28 (gdb) n
29      bne    loop
30 (gdb) n
31      loop:  ldrsb  r1, [r0]
32 (gdb) n
33      add    r2, r2, r1    @ r2 = r2 + r1
34 (gdb) n
35      add    r0, r0, #1    @ point to next
36 (gdb) n
37      subs   r3, r3, #1    @ decrement counter
38 (gdb) n
39      bne    loop
40 (gdb) i r
41      r0      0x200af      131247
42      r1      0x13        19
43      r2      0x10        16
44      r3      0x0         0
45      r4      0x0         0
46      r5      0x0         0
47      r6      0x0         0
48      r7      0x0         0
49      r8      0x0         0
50      r9      0x0         0
51      r10     0x0         0
52      r11     0x0         0
53      r12     0x0         0
54      sp      0xfffff000  0
55      lr      0xfffff000  0
56      pc      0x10000     0x10000 <loop+15>
57      cpsr    0x20000010  536870928
58      timer   0x0         0
59 (gdb)

```

4. Lowest Progs

- Describe how the program works. Make sure you describe the pointer and counter function. For each of 2-5, make sure you include flags in your discussion of "how does it work"
- What is the beq instruction and how does it work?

The BEQ tests if the z flag is set to 1, indicating the previous arithmetic resulted in zero. If it is the branch is taken, if not the instruction following BEQ is taken.

- What is the cmp instruction and how does it work?

CMP subtracts the second operand from the first operand but does not store the result, only updates the appropriate flags based on the results.

- What is the movlt instruction and how does it work?

MOVLT checks the negative flag and overflow flag. If negative is set but overflow isn't the MOVLT instruction is taken. Otherwise it is skipped.

- What is the b instruction and how does it work?

The B instruction causes the program to jump to a specified target address.

- Change the program by changing the conditional mov instruction so that the program finds the largest of the signed numbers. Show your final code.

The screenshot shows the Visual Studio Code interface with the following components:

- Explorer:** A file tree on the left showing a project named '4_lowest_prog5.2.s'. It contains several files including 'Deliverable_Pics', '1_flags_ex5.13.o', '2_flagsfixed_ex5.13.o', '3_flagsfixed_ex5.13.o', '4_lowest_prog5.2.o', '5_lowest_prog5.2.s', and 'makefile'.
- Editor:** The main window displays the assembly code for '4_lowest_prog5.2.s'. The code is as follows:


```

14
15
16 .text
17 .global _start
18 _start:
19     ldr r0, =sign_dat
20     mov r3, #0
21     ldrsb r2, [r0] @ bring first number into r2 and sign extend it
22 loop:
23     add r0, r0, #1 @ point to next
24     subs r3, r3, #1 @ decrement counter
25     beq done @ if r3 is zero, done
26     ldrsb r1, [r0] @ bring next number into r1 and sign extend it
27     cmp r1, r2 @ compare r1 and r2
28     movgt r2, r1 @ if r1 is greater, keep it in r2
29     b loop

```
- Debugger:** The bottom panel shows the execution state. The 'TERMINAL' tab is active, displaying the following output:


```

(gdb) n
28      b      loop
(gdb) n
23      add   r0, r0, #1 @ point to next
(gdb) n
24      subs  r3, r3, #1 @ decrement counter
(gdb) n
25      beq   done @ if r3 is zero, done
(gdb) n
26      ldrsb r1, [r0] @ bring next number into r1 and sign extend it
(gdb) n
27      cmp   r1, r2 @ compare r1 and r2
(gdb) n
28      movgt r2, r1 @ if r1 is greater, keep it in r2
(gdb) n
29      b     loop
(gdb) i r
r0      0x200006      131254
r1      0x13         19
r2      0x13         19
r3      0x7          7
r4      0x0          0
r5      0x0          0
r6      0x0          0
r7      0x0          0
r8      0x0          0
r9      0x0          0
r10     0x0          0
r11     0x0          0
r12     0x0          0
sp      0xfffff0e0    0xfffff0e0
lr      0x0          0
pc      0x100008      0x100008 <loop+24>
cpsr    0x20000010    536870928
(gdb)

```

5. Match

- Explain how the program works. Make sure you describe the pointer/counters
- What is the cmn instruction and how does it work?

The CMN is similar to CMP in that it subtracts two operands and updates the register flags without saving the result. The updated flags are N, Z, V, and C flags.

- What does the program do if a match is not found?

If a match is not found the programs increments a counter and loops back to check. When a match is found the program ends.

- Change one instruction only so that the program finds the match of the number (not the negative of the number). Show your program.

