

UNIVERSITY OF VICTORIA
EXAMINATIONS FALL 2012

**CSC 230 - Introduction to Computer Architecture and
 Assembly Language - A01 CRN# 12749 and A02 CRN# 12750**

STUDENT NUMBER: _____

TIME: 3 hours

INSTRUCTOR: M. Serra

TOTAL MARKS: 102

TO BE ANSWERED ON THE PAPER

Question No.	Value	Mark	Question No.	Value	Mark
1	6		11	2	
2	6		12	4	
3	5		13	4	
4	4		14	4	
5	4		15	4	
6	4		16	6	
7	6		17	8	
8	4		18	8	
9	4		19	13	
10	6		TOTAL	102	

INSTRUCTIONS:

1. STUDENTS MUST COUNT THE NUMBER OF PAGES IN THIS EXAMINATION PAPER BEFORE BEGINNING TO WRITE, AND REPORT ANY DISCREPANCY IMMEDIATELY TO THE INVIGILATOR
2. This examination paper consists of 16 pages including this cover page.
3. No aids are permitted. However, a handout describing the ARM instruction set is provided for your use.
4. The marks assigned to each question are shown within square brackets. Partial marks are available for all questions.
5. Please be precise but brief, and use point form where appropriate.
6. It is strongly recommended that you read the entire exam through from beginning to end before beginning to answer the questions.

Question 1. [6] Fill in the table below with the appropriate information about the instructions. The column "Bus used?" refers ONLY to the execution phase, not the fetch phase.

Instruction	Addressing Mode of Source Operand	Bus used ?	What it does (be precise)
TT r1, #1	register direct	none	test if r1 is 0 or negative
LR r1, [r2, r3, lsl #2]!	register direct data direct	control data address	loads register
MV r1, r2, asr #4	register direct	control data address	copies r2 into r1

Question 2. [6] You have a system with Virtual memory, a DMA, a Page Table, a TLB, RAM memory, disks, and two levels of cache L1 and L2, with L1 being further subdivided into an L1 Data cache and an L1 Instruction cache.

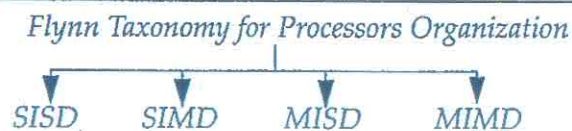
(a) [3] The CPU wants to access some data and the data is in L2, but not in L1. Describe the series of events precisely and concisely.

- request is sent to L1 and TLB in parallel
- data not in L1 so L2 is checked
- L2 returns the data

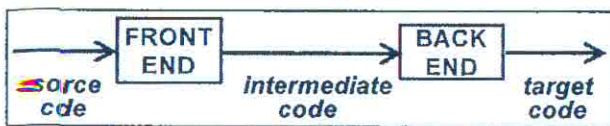
[3] Repeat, assuming the data is in RAM memory and not in the L1 or L2 caches.

- request sent to L1 and TLR
- data not in L1 so L2 checked
- at same time Page table checked
- data not in L2
- data found in main memory or disk using Page table
- data returned

Question 3. [5] (a) [4] The top portion of the Flynn taxonomy is shown below. State the *definitions* only for the four categories.



(b) [1] Describe the differences between SIMD and MIMD briefly.

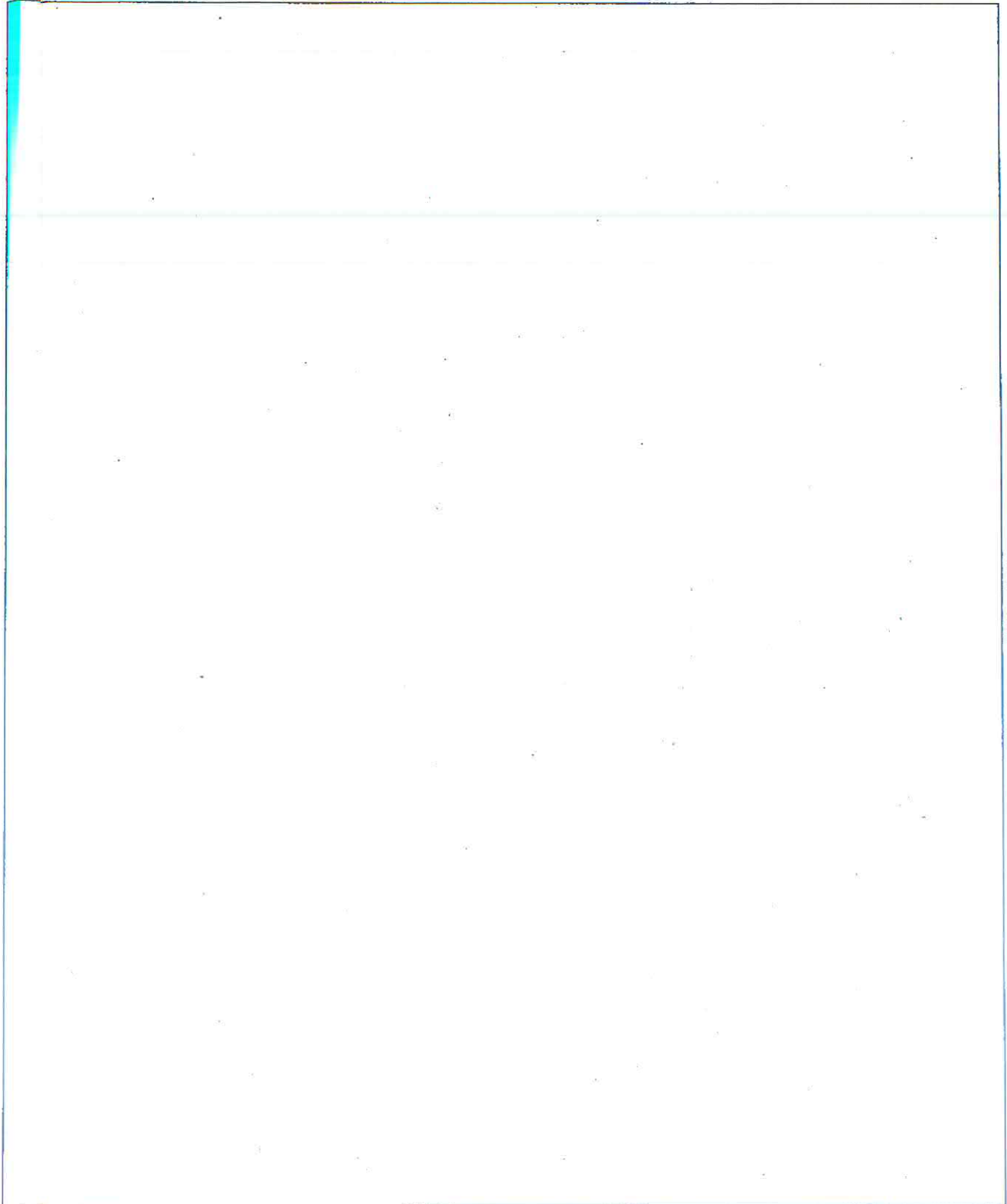


Question 4. [4] (a) [3] The figure shows the structure of a compiler decomposed into a "Front End" and a "Back End" with *Intermediate Code* generated in between. Explain briefly, in point

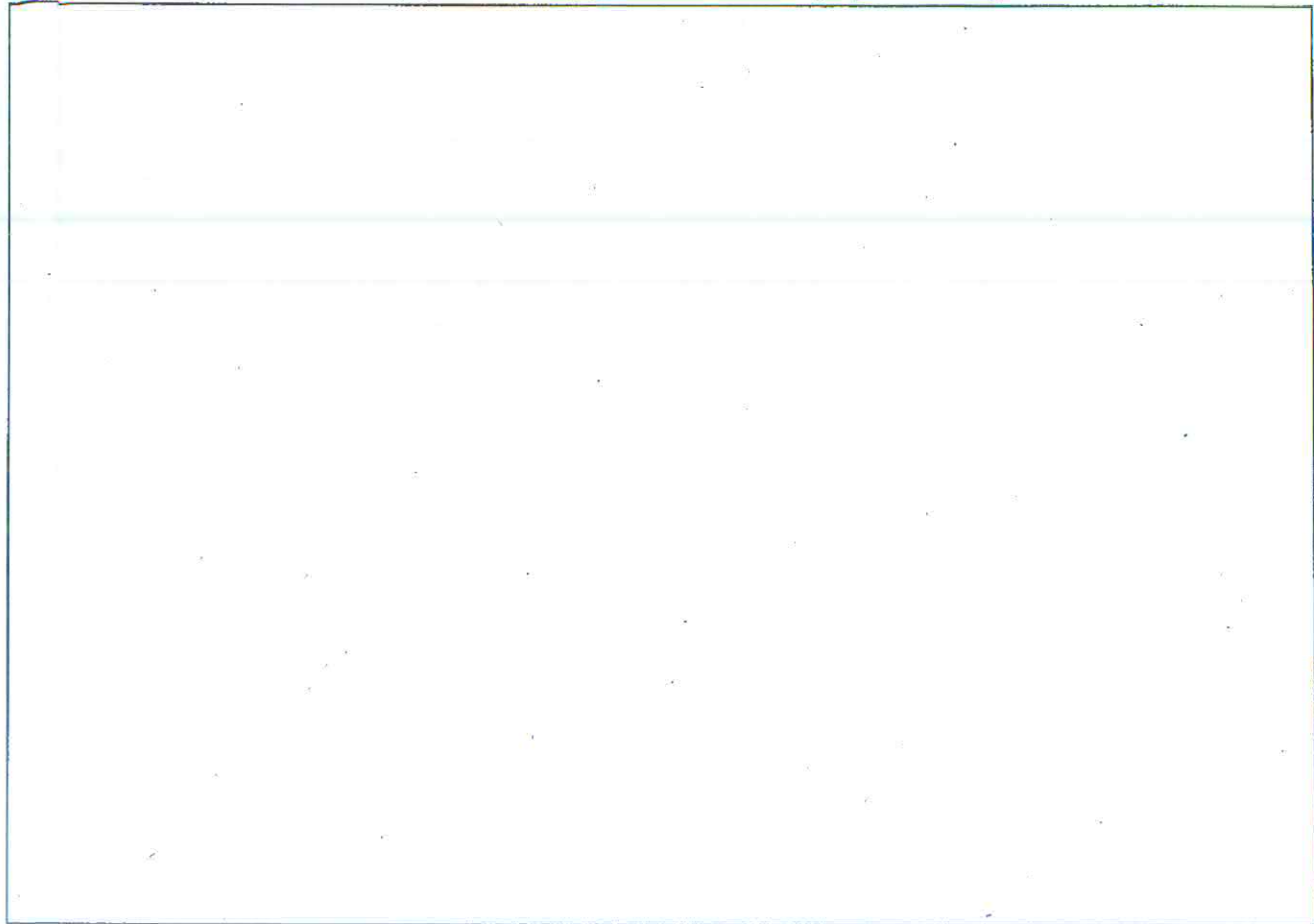
form, what the phases are in the front end, that is, when the high level source code is translated into intermediate code. In an interview situation, you should not take more than 2 minutes; be similarly concise and precise.

(b) [1] In the explanation above, somewhere, the notion of a *Symbol Table* should have surfaced. Describe in more depth what the function of the Symbol Table is within the process.

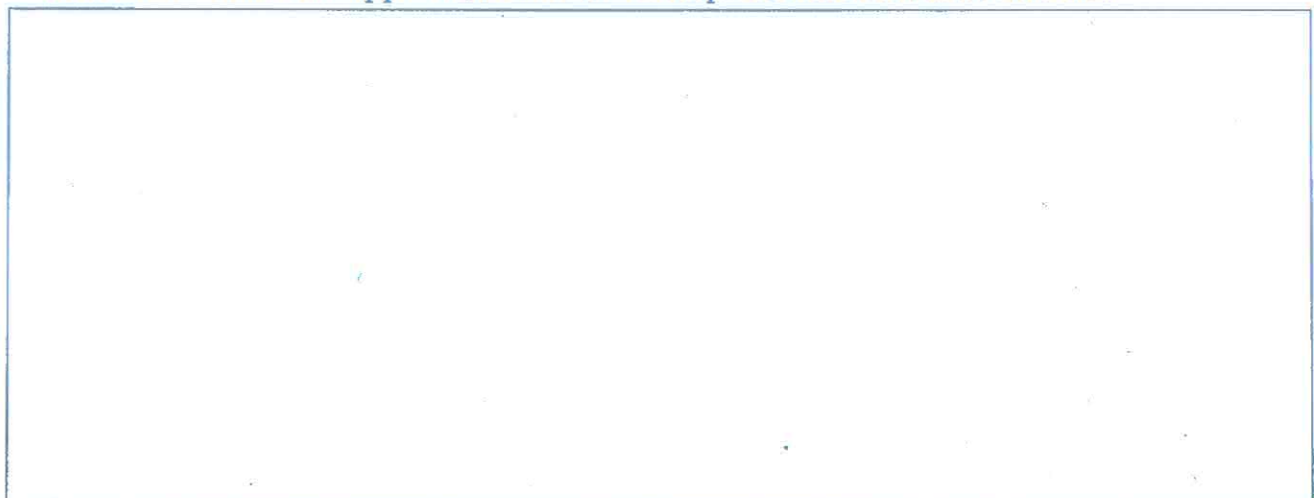
Question 5. [4] Dynamic linking is an important concept and the differences with static linking have been discussed. Draw *two* diagrams to summarize the elements of the processes of dynamic and static linking. Diagrams only with the elements involved are required here, not a complete explanation of how dynamic linking works.

A large empty rectangular box with a thin black border, intended for the student to draw two diagrams summarizing the elements of dynamic and static linking processes.

Question 6. [4] (a) [3] An interrupt signal is sent to the processor from a device interface. Summarize the possible series of events which follow by drawing a flowchart or a process chart or some sort of clear and precise diagram. For each step you should not write a long detailed explanation. However you should attach a short label/definition to each step that makes it clear that you know and understand what is supposed to happen in the interrupt process sequence of steps.



(b) [1] After an exception or interrupt has been processed, it may be the case that normal processing can resume within the application. Give one example in which this is not the case.



Question 7. [6] The following initialization loop is included in the C code that solves a vector processing algorithm:

```
for (i=0; i++; i<50)
    V[i] = -85 * i;
```

You are in the process of converting this algorithm to ARM assembly language. In the data section, 50 words have been reserved for the vector V and, in the code section, the address of V has been loaded into R0 (as shown below). This code is in the middle of a complex routine that uses all registers except R7 and R9. Using these two available registers and no more than six ARM instructions, write the initialization loop. (Six instructions is optimal, however answers with more instructions are also accepted with minimal penalty).

```
@ === Other code comes before this line
ldr R0, =V          @R0 = Address of vector V
ldr R7, 0

loop: muli R9, R7, -85
      st R0, R9
      linc R0
      inc R7
      cpi R7, 50
      bgt loop

@ === Other code comes after this line
.data
V: .skip 200
.end
```

Question 8. [4] Fill in the right column of the table with your answers.

1. The range of decimal values that can be represented as unsigned 16 bit integers is (answer in decimal):	$(2^{15}, 2^{16}-1)$
2. Given 8 binary bits, the largest positive integer that can be represented using a 2's complement representation is (answer in decimal):	$2^8-1 = 255$
3. Given 16 binary bits, the largest negative integer that can be represented using a signed magnitude representation is:	-2^{15}
4. The hexadecimal value FFE corresponds to the 12-bit binary:	1111 1111 1110
5. The hexadecimal value FFE viewed as a 12-bit integer in a 2's complement context, corresponds to the decimal:	-2
6. Convert the decimal integer -23 to 8-bit binary in 2's complement	1110 1001
7. Convert the decimal integer -23 to 3-digit hexadecimal in 2's complement	FE9
8. The 2-digit hexadecimal quantity AA is even and negative (TRUE/FALSE/NEITHER)	True both even and negative

5. 1111 1111 1110 0000 0000 0001
 6. 23 0001 0111 1110 1000
 7. 1111 1110 1001
 8. 1010 1010
 9. multi 0101 0101
 10. 0101 0110
 11. 0101 0110
 12. 0101 0110
 13. 0101 0110
 14. 0101 0110
 15. 0101 0110
 16. 0101 0110
 17. 0101 0110
 18. 0101 0110
 19. 0101 0110
 20. 0101 0110
 21. 0101 0110
 22. 0101 0110
 23. 0101 0110
 24. 0101 0110
 25. 0101 0110
 26. 0101 0110
 27. 0101 0110
 28. 0101 0110
 29. 0101 0110
 30. 0101 0110
 31. 0101 0110
 32. 0101 0110
 33. 0101 0110
 34. 0101 0110
 35. 0101 0110
 36. 0101 0110
 37. 0101 0110
 38. 0101 0110
 39. 0101 0110
 40. 0101 0110
 41. 0101 0110
 42. 0101 0110
 43. 0101 0110
 44. 0101 0110
 45. 0101 0110
 46. 0101 0110
 47. 0101 0110
 48. 0101 0110
 49. 0101 0110
 50. 0101 0110
 51. 0101 0110
 52. 0101 0110
 53. 0101 0110
 54. 0101 0110
 55. 0101 0110
 56. 0101 0110
 57. 0101 0110
 58. 0101 0110
 59. 0101 0110
 60. 0101 0110
 61. 0101 0110
 62. 0101 0110
 63. 0101 0110
 64. 0101 0110
 65. 0101 0110
 66. 0101 0110
 67. 0101 0110
 68. 0101 0110
 69. 0101 0110
 70. 0101 0110
 71. 0101 0110
 72. 0101 0110
 73. 0101 0110
 74. 0101 0110
 75. 0101 0110
 76. 0101 0110
 77. 0101 0110
 78. 0101 0110
 79. 0101 0110
 80. 0101 0110
 81. 0101 0110
 82. 0101 0110
 83. 0101 0110
 84. 0101 0110
 85. 0101 0110
 86. 0101 0110
 87. 0101 0110
 88. 0101 0110
 89. 0101 0110
 90. 0101 0110
 91. 0101 0110
 92. 0101 0110
 93. 0101 0110
 94. 0101 0110
 95. 0101 0110
 96. 0101 0110
 97. 0101 0110
 98. 0101 0110
 99. 0101 0110
 100. 0101 0110
 101. 0101 0110
 102. 0101 0110
 103. 0101 0110
 104. 0101 0110
 105. 0101 0110
 106. 0101 0110
 107. 0101 0110
 108. 0101 0110
 109. 0101 0110
 110. 0101 0110
 111. 0101 0110
 112. 0101 0110
 113. 0101 0110
 114. 0101 0110
 115. 0101 0110
 116. 0101 0110
 117. 0101 0110
 118. 0101 0110
 119. 0101 0110
 120. 0101 0110
 121. 0101 0110
 122. 0101 0110
 123. 0101 0110
 124. 0101 0110
 125. 0101 0110
 126. 0101 0110
 127. 0101 0110
 128. 0101 0110
 129. 0101 0110
 130. 0101 0110
 131. 0101 0110
 132. 0101 0110
 133. 0101 0110
 134. 0101 0110
 135. 0101 0110
 136. 0101 0110
 137. 0101 0110
 138. 0101 0110
 139. 0101 0110
 140. 0101 0110
 141. 0101 0110
 142. 0101 0110
 143. 0101 0110
 144. 0101 0110
 145. 0101 0110
 146. 0101 0110
 147. 0101 0110
 148. 0101 0110
 149. 0101 0110
 150. 0101 0110
 151. 0101 0110
 152. 0101 0110
 153. 0101 0110
 154. 0101 0110
 155. 0101 0110
 156. 0101 0110
 157. 0101 0110
 158. 0101 0110
 159. 0101 0110
 160. 0101 0110
 161. 0101 0110
 162. 0101 0110
 163. 0101 0110
 164. 0101 0110
 165. 0101 0110
 166. 0101 0110
 167. 0101 0110
 168. 0101 0110
 169. 0101 0110
 170. 0101 0110
 171. 0101 0110
 172. 0101 0110
 173. 0101 0110
 174. 0101 0110
 175. 0101 0110
 176. 0101 0110
 177. 0101 0110
 178. 0101 0110
 179. 0101 0110
 180. 0101 0110
 181. 0101 0110
 182. 0101 0110
 183. 0101 0110
 184. 0101 0110
 185. 0101 0110
 186. 0101 0110
 187. 0101 0110
 188. 0101 0110
 189. 0101 0110
 190. 0101 0110
 191. 0101 0110
 192. 0101 0110
 193. 0101 0110
 194. 0101 0110
 195. 0101 0110
 196. 0101 0110
 197. 0101 0110
 198. 0101 0110
 199. 0101 0110
 200. 0101 0110
 201. 0101 0110
 202. 0101 0110
 203. 0101 0110
 204. 0101 0110
 205. 0101 0110
 206. 0101 0110
 207. 0101 0110
 208. 0101 0110
 209. 0101 0110
 210. 0101 0110
 211. 0101 0110
 212. 0101 0110
 213. 0101 0110
 214. 0101 0110
 215. 0101 0110
 216. 0101 0110
 217. 0101 0110
 218. 0101 0110
 219. 0101 0110
 220. 0101 0110
 221. 0101 0110
 222. 0101 0110
 223. 0101 0110
 224. 0101 0110
 225. 0101 0110
 226. 0101 0110
 227. 0101 0110
 228. 0101 0110
 229. 0101 0110
 230. 0101 0110
 231. 0101 0110
 232. 0101 0110
 233. 0101 0110
 234. 0101 0110
 235. 0101 0110
 236. 0101 0110
 237. 0101 0110
 238. 0101 0110
 239. 0101 0110
 240. 0101 0110
 241. 0101 0110
 242. 0101 0110
 243. 0101 0110
 244. 0101 0110
 245. 0101 0110
 246. 0101 0110
 247. 0101 0110
 248. 0101 0110
 249. 0101 0110
 250. 0101 0110
 251. 0101 0110
 252. 0101 0110
 253. 0101 0110
 254. 0101 0110
 255. 0101 0110
 256. 0101 0110
 257. 0101 0110
 258. 0101 0110
 259. 0101 0110
 260. 0101 0110
 261. 0101 0110
 262. 0101 0110
 263. 0101 0110
 264. 0101 0110
 265. 0101 0110
 266. 0101 0110
 267. 0101 0110
 268. 0101 0110
 269. 0101 0110
 270. 0101 0110
 271. 0101 0110
 272. 0101 0110
 273. 0101 0110
 274. 0101 0110
 275. 0101 0110
 276. 0101 0110
 277. 0101 0110
 278. 0101 0110
 279. 0101 0110
 280. 0101 0110
 281. 0101 0110
 282. 0101 0110
 283. 0101 0110
 284. 0101 0110
 285. 0101 0110
 286. 0101 0110
 287. 0101 0110
 288. 0101 0110
 289. 0101 0110
 290. 0101 0110
 291. 0101 0110
 292. 0101 0110
 293. 0101 0110
 294. 0101 0110
 295. 0101 0110
 296. 0101 0110
 297. 0101 0110
 298. 0101 0110
 299. 0101 0110
 300. 0101 0110
 301. 0101 0110
 302. 0101 0110
 303. 0101 0110
 304. 0101 0110
 305. 0101 0110
 306. 0101 0110
 307. 0101 0110
 308. 0101 0110
 309. 0101 0110
 310. 0101 0110
 311. 0101 0110
 312. 0101 0110
 313. 0101 0110
 314. 0101 0110
 315. 0101 0110
 316. 0101 0110
 317. 0101 0110
 318. 0101 0110
 319. 0101 0110
 320. 0101 0110
 321. 0101 0110
 322. 0101 0110
 323. 0101 0110
 324. 0101 0110
 325. 0101 0110
 326. 0101 0110
 327. 0101 0110
 328. 0101 0110
 329. 0101 0110
 330. 0101 0110
 331. 0101 0110
 332. 0101 0110
 333. 0101 0110
 334. 0101 0110
 335. 0101 0110
 336. 0101 0110
 337. 0101 0110
 338. 0101 0110
 339. 0101 0110
 340. 0101 0110
 341. 0101 0110
 342. 0101 0110
 343. 0101 0110
 344. 0101 0110
 345. 0101 0110
 346. 0101 0110
 347. 0101 0110
 348. 0101 0110
 349. 0101 0110
 350. 0101 0110
 351. 0101 0110
 352. 0101 0110
 353. 0101 0110
 354. 0101 0110
 355. 0101 0110
 356. 0101 0110
 357. 0101 0110
 358. 0101 0110
 359. 0101 0110
 360. 0101 0110
 361. 0101 0110
 362. 0101 0110
 363. 0101 0110
 364. 0101 0110
 365. 0101 0110
 366. 0101 0110
 367. 0101 0110
 368. 0101 0110
 369. 0101 0110
 370. 0101 0110
 371. 0101 0110
 372. 0101 0110
 373. 0101 0110
 374. 0101 0110
 375. 0101 0110
 376. 0101 0110
 377. 0101 0110
 378. 0101 0110
 379. 0101 0110
 380. 0101 0110
 381. 0101 0110
 382. 0101 0110
 383. 0101 0110
 384. 0101 0110
 385. 0101 0110
 386. 0101 0110
 387. 0101 0110
 388. 0101 0110
 389. 0101 0110
 390. 0101 0110
 391. 0101 0110
 392. 0101 0110
 393. 0101 0110
 394. 0101 0110
 395. 0101 0110
 396. 0101 0110
 397. 0101 0110
 398. 0101 0110
 399. 0101 0110
 400. 0101 0110
 401. 0101 0110
 402. 0101 0110
 403. 0101 0110
 404. 0101 0110
 405. 0101 0110
 406. 0101 0110
 407. 0101 0110
 408. 0101 0110
 409. 0101 0110
 410. 0101 0110
 411. 0101 0110
 412. 0101 0110
 413. 0101 0110
 414. 0101 0110
 415. 0101 0110
 416. 0101 0110
 417. 0101 0110
 418. 0101 0110
 419. 0101 0110
 420. 0101 0110
 421. 0101 0110
 422. 0101 0110
 423. 0101 0110
 424. 0101 0110
 425. 0101 0110
 426. 0101 0110
 427. 0101 0110
 428. 0101 0110
 429. 0101 0110
 430. 0101 0110
 431. 0101 0110
 432. 0101 0110
 433. 0101 0110
 434. 0101 0110
 435. 0101 0110
 436. 0101 0110
 437. 0101 0110
 438. 0101 0110
 439. 0101 0110
 440. 0101 0110
 441. 0101 0110
 442. 0101 0110
 443. 0101 0110
 444. 0101 0110
 445. 0101 0110
 446. 0101 0110
 447. 0101 0110
 448. 0101 0110
 449. 0101 0110
 450. 0101 0110
 451. 0101 0110
 452. 0101 0110
 453. 0101 0110
 454. 0101 0110
 455. 0101 0110
 456. 0101 0110
 457. 0101 0110
 458. 0101 0110
 459. 0101 0110
 460. 0101 0110
 461. 0101 0110
 462. 0101 0110
 463. 0101 0110
 464. 0101 0110
 465. 0101 0110
 466. 0101 0110
 467. 0101 0110
 468. 0101 0110
 469. 0101 0110
 470. 0101 0110
 471. 0101 0110
 472. 0101 0110
 473. 0101 0110
 474. 0101 0110
 475. 0101 0110
 476. 0101 0110
 477. 0101 0110
 478. 0101 0110
 479. 0101 0110
 480. 0101 0110
 481. 0101 0110
 482. 0101 0110
 483. 0101 0110
 484. 0101 0110
 485. 0101 0110
 486. 0101 0110
 487. 0101 0110
 488. 0101 0110
 489. 0101 0110
 490. 0101 0110
 491. 0101 0110
 492. 0101 0110
 493. 0101 0110
 494. 0101 0110
 495. 0101 0110
 496. 0101 0110
 497. 0101 0110
 498. 0101 0110
 499. 0101 0110
 500. 0101 0110
 501. 0101 0110
 502. 0101 0110
 503. 0101 0110
 504. 0101 0110
 505. 0101 0110
 506. 0101 0110
 507. 0101 0110
 508. 0101 0110
 509. 0101 0110
 510. 0101 0110
 511. 0101 0110
 512. 0101 0110
 513. 0101 0110
 514. 0101 0110
 515. 0101 0110
 516. 0101 0110
 517. 0101 0110
 518. 0101 0110
 519. 0101 0110
 520. 0101 0110
 521. 0101 0110
 522. 0101 0110
 523. 0101 0110
 524. 0101 0110
 525. 0101 0110
 526. 0101 0110
 527. 0101 0110
 528. 0101 0110
 529. 0101 0110
 530. 0101 0110
 531. 0101 0110
 532. 0101 0110
 533. 0101 0110
 534. 0101 0110
 535. 0101 0110
 536. 0101 0110
 537. 0101 0110
 538. 0101 0110
 539. 0101 0110
 540. 0101 0110
 541. 0101 0110
 542. 0101 0110
 543. 0101 0110
 544. 0101 0110
 545. 0101 0110
 546. 0101 0110
 547. 0101 0110
 548. 0101 0110
 549. 0101 0110
 550. 0101 0110
 551. 0101 0110
 552. 0101 0110
 553. 0101 0110
 554. 0101 0110
 555. 0101 0110
 556. 0101 0110
 557. 0101 0110
 558. 0101 0110
 559. 0101 0110
 560. 0101 0110
 561. 0101 0110
 562. 0101 0110
 563. 0101 0110
 564. 0101 0110
 565. 0101 0110
 566. 0101 0110
 567. 0101 0110
 568. 0101 0110
 569. 0101 0110
 570. 0101 0110
 571. 0101 0110
 572. 0101 0110
 573. 0101 0110
 574. 0101 0110
 575. 0101 0110
 576. 0101 0110
 577. 0101 0110
 578. 0101 0110
 579. 0101 0110
 580. 0101 0110
 581. 0101 0110
 582. 0101 0110
 583. 0101 0110
 584. 0101 0110
 585. 0101 0110
 586. 0101 0110
 587. 0101 0110
 588. 0101 0110
 589. 0101 0110
 590. 0101 0110
 591. 0101 0110
 592. 0101 0110
 593. 0101 0110
 594. 0101 0110
 595. 0101 0110
 596. 0101 0110
 597. 0101 0110
 598. 0101 0110
 599. 0101 0110
 600. 0101 0110
 601. 0101 0110
 602. 0101 0110
 603. 0101 0110
 604. 0101 0110
 605. 0101 0110
 606. 0101 0110
 607. 0101 0110
 608. 0101 0110
 609. 0101 0110

$$64KB = 2^6 \cdot 2^{10} = 2^{16} \text{ bytes}$$

$$\text{Total } 2^{16} + 2^4 + 2^4 + 2^4 + 2^4 = 65600$$

Question 9. [4] A computer system has a RAM containing 64K bytes, each of which needs its own distinct address. Moreover it has 4 peripherals and they each require 2^4 distinct addresses in order to interface properly.

(a) How many distinct addresses in total are necessary in this system? Write the total number in the centre of the expression below on the left. (Feel free to leave it as sum of powers).

$$2^{\boxed{16}} < 65600 < 2^{\boxed{17}}$$

total number of addresses here

(b) Place the number of addresses just computed between the appropriate powers of two in the expression on the left, by writing the correct exponent in the boxes (for example, if the result were 18, you would have $2^4 < 18 < 2^5$ by writing the exponents 4 and 5).

(c) How many lines does an address bus for this system require, given that it must be able to carry all the needed values for the addresses? 17

Question 10. [6] Consider two different machines, with two different instruction sets, and with the same clock rate of 200MHz.¹ The following measurements are recorded on the two machines running a given set of benchmark programs:

Instruction type	Instruction count (millions)	Cycles per instruction
Machine A		
Arithmetic and logic	8	1
Load and store	4	3
Branch	4	2
Others	4	3
Machine B		
Arithmetic and logic	10	1
Load and store	8	2
Branch	2	4
Others	4	3

(a) [2] Determine the effective CPI for each machine (show your calculations).

1. Hz is cycles/sec, MHz = (cycles/sec) $\times 10^6$, 1 second = 10^9 ns.

b [2] Compute the execution time in ns of the benchmarks for each machine (show your work).

c [2] A common measure of performance for a processor is the rate at which instructions are executed, expressed as millions of instructions per second (MIPS), referred to as the MIPS rate. We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

State the MIPS rate for each machine above (at least show the complete equation for the calculation, even if you cannot handle the computation).

Question 11. [2] While browsing at a computer store, you overhear a customer asking a salesperson what is the fastest computer in the store that can be purchased. The employee replies: "You are standing and looking at the moment at our Macintoshes. The fastest Mac here runs at a clock speed of 1.2 GHz. If you really want the fastest machine, you should buy our 2.4 GHz Intel Pentium IV instead." Is the salesperson correct? What would you say to help the customer?

Question 12. [4] For a system with two levels of cache, define the following:

T_C = first-level cache access time;

T_{C2} = second-level cache access time;

T_M = memory access time;

H_1 = first-level cache hit ratio;

H_2 = combined first/second level cache hit ratio.

Provide an equation for T , the total cache access time for a read operation.

$$T = H_1 \times T_{C1} + (H_2 - H_1) \times T_{C2} + (1 - H_2) \times T_M$$

Question 13. [4] When evaluating the performance of a pipeline, we calculate the possible speedup as the ratio:

$$\text{Speedup} = \frac{T_{\text{serial}}}{T_{\text{pipeline}}} = \frac{m \times N}{m + N - 1}$$

An old processor (very similar to the Intel 8088) consists of a bus interface unit (BIU) and an execution unit (EU), which form a 2-stage pipeline. The BIU fetches instructions into a 4-byte instruction queue. The BIU also participates in address calculation, fetches operands, and writes results in memory as requested by the EU. If no such requests are outstanding and the bus is free, the BIU fills any vacancies in the instruction queue. When the EU completes execution of an instruction, it passes any results to the BIU (destined for memory or I/O) and proceeds to the next instruction.

(a) [2] Suppose the tasks performed by the BIU and the EU take about equal time. By what factor does pipelining improve the performance of this processor? Ignore any consideration of branch instructions. Show your work.

It fetches the next instruction while the EU processes the current instruction
 speed up = 2

(b) [2] Repeat the evaluation assuming that the EU takes twice as long as the BIU. Show your work.

Question 14. [4] A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 20ns are required to access it. If it is in main memory but not in the cache, 60 ns are needed to load it into the cache, and then the reference is started again. If the word is not in main memory, 12 ms are required to fetch the word from disk, followed by 60 ns to copy it to the cache, and then the reference is started again. The cache hit ratio is 0.95 and the main memory hit ratio is 0.9. What is the average time in ns required to access a referenced word on this system (show your work in detail)?¹

$$T_{avg} = 0.95 \times 20ns + 0.9 \times [60ns + 20ns] + (1 - 0.9 \times 0.95) [12ms + 60ns] + 20ns$$

↑ copy to cache
 ↑ reference
 ↑ fetch from disk
 ↑ copy to cache
 ↑ reference

Question 15. [4] (a) [1] Describe briefly the main function of a DMA.

(b) [1] State what peripheral interfaces are.

1. 1 s = 1,000 ms = 1,000,000,000 ns. Thus 1 ms = 10^6 ns

(c) Explain "polling" as a synchronization method for communication between a processor and peripherals through their interfaces.

Question 16. [6] Make comparisons between caches and virtual memory with respect to: purpose, unit of data transfer, and method of implementation:

	Cache	Virtual Memory
purpose	speed	expand memory
unit of data transfer	line / block	page
method of implementation: hardware or software or both	hardware	both

Question 17. [8] Consider a memory of 64K words ($= 2^{16}$ words), addressable by 16-bit addresses, to be viewed as 4K blocks ($= 2^{12}$ blocks) of 16 words each (2^{12} blocks \times 2^4 words $= 2^{16}$ words). Consider also a cache consisting of 128 blocks of 16 words each, for a total of 2048 (2K) words. Assume that a direct mapping configuration is used for the cache organization, as shown in one part of figure 1.

- (a)[1] Describe precisely how and where an arbitrary 16 word block K from memory is loaded into the cache by summarizing the general strategy.

block k is loaded into address $k \bmod \# \text{ of blocks in cache}$
 address $k = k \% 128$

- (b)[3] Show that you understand the strategy by giving the cache block numbers assigned to the following *sequence* of blocks fetched from memory – the blocks are fetched in the order as written from left to right (there might be collisions).

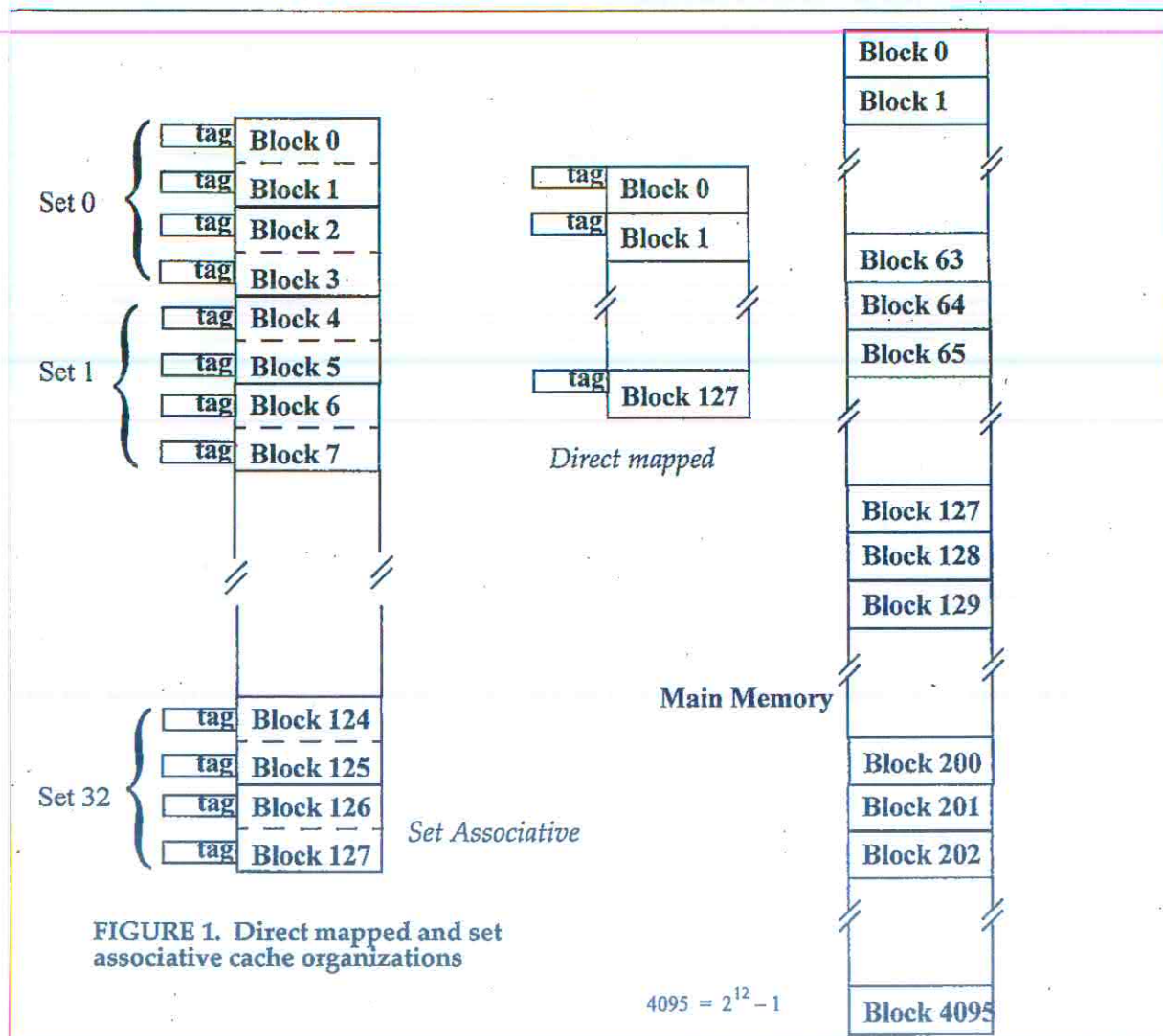
Memory Block #	0	1	65	200	202	128	129
Cache Block #	0	1	65	72	74	0	1

- (c)[1] Assume now that a set associative mapping organization is used for the same cache, seen as 32 sets of 4 blocks each. Describe precisely how the mapping is configured differently from direct mapping, again looking also at the left part of figure 1.

cache set $\# = k \% \# \text{ of sets (32 in this case)}$
 cache block $\#$: which ever one is open
 with set associative strategy the k is for sure in one set but can be in any block in that set

- (d)[3] Repeat part (b) from above assuming set associative mapping for the same cache, seen as 32 sets of 4 blocks each. Give the cache set and block numbers assigned to the following *sequence* of blocks fetched from memory – the blocks are fetched in the order as written from left to right (there might be collisions).

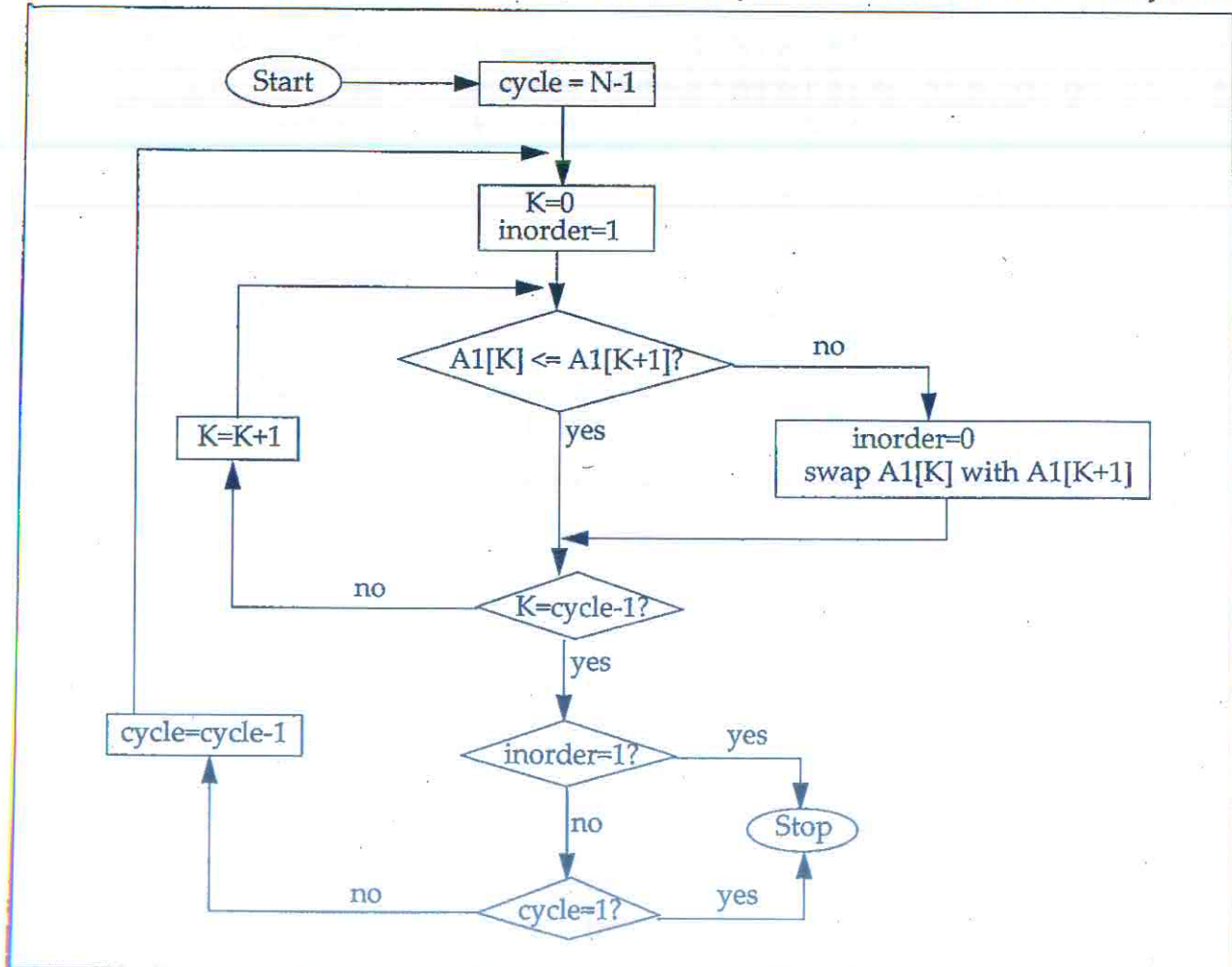
Memory Block #	0	1	65	200	202	128	129
Cache Set #	0	1	65	72	74	0	1
Cache Block #	0	0	0	0	0	1	1



Question 18. [8] Give brief definition for the following concepts related to virtual memory:

Concept/Term	Definition
Page	a continuous block of virtual memory (several lines of memory together)
Page fault	when a page is not found in the page table
TLB (Translation Lookaside Buffer)	cache cache for recent address translation
MMU (Memory Management Unit)	manages the reading and writing from the main memory to disk

Question 19. [13] Write a procedure to sort an array of 32-bit integers using the bubble sort method. The array is called ARR1 (declared by main in .DATA) and its size is stored in ARCNT. Assume that the main routine calls an external function INIT which, given the address of the array, returns it filled with data and with its current size in R0. The main program passes the address of ARR and its size from ARCNT as parameters to the procedure BubbleSort in R1 and R0 respectively. You are supposed to write the whole program. The flowchart for the algorithm of BubbleSort is given below. Get it manually first on a small example to understand well what you are supposed to be coding. In the flowchart, N denotes the number of elements in the array and A1 is the label for the array itself.¹



1. Should you think there is any problem with the algorithm and/or the flowchart, disregard for the moment and just proceed to code as shown.

Operand 2	
Immediate value	#<immed_8>
Logical shift left immediate	Rm, LSL #<immed_5>
Logical shift right immediate	Rm, LSR #<immed_5>
Arithmetic shift right immediate	Rm, ASR #<immed_5>
Rotate right immediate	Rm, ROR #<immed_5>
Register	Rm
Rotate right extended	Rm, RRX
Logical shift left register	Rm, LSL Rs
Logical shift right register	Rm, LSR Rs
Arithmetic shift right register	Rm, ASR Rs
Rotate right register	Rm, ROR Rs

Addressing Mode 4 - Multiple Data Transfer			
Block load		Stack pop	
IA	Increment After	FD	Full Descending
IB	Increment Before	ED	Empty Descending
DA	Decrement After	FA	Full Ascending
DB	Decrement Before	EA	Empty Ascending
Block store		Stack push	
IA	Increment After	EA	Empty Ascending
IB	Increment Before	FA	Full Ascending
DA	Decrement After	ED	Empty Descending
DB	Decrement Before	FD	Full Descending

Condition Field	
EQ	Equal
NE	Not equal
CS	Carry Set
CC	Carry clear
MI	Negative
PL	Positive or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL	Always

Dec	Bin	Hex
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F

D	BIN	H	D	BIN	H	D	BIN	H
0	00000000	00	4	00000100	04	8	00001000	08
1	00000001	01	5	00000101	05	9	00001001	09
2	00000010	02	6	00000110	06	10	00001010	0A
3	00000011	03	7	00000111	07	11	00001011	0B
						12	00001100	0C
						13	00001101	0D
						14	00001110	0E
						15	00001111	0F

Operation	Assembler	Action
Move	MOV{S} Rd, <Oprnd2>	Rd := Oprnd2 {CPSR}
	MVN{S} Rd, <Oprnd2>	Rd := NOT Oprnd2 {CPSR}
Arithmetic	ADD{S} Rd, Rn, <Oprnd2>	Rd := Rn + Oprnd2 {CPSR}
	ADC{S} Rd, Rn, <Oprnd2>	Rd := Rn + Oprnd2 + Carry {CPSR}
	SUB{S} Rd, Rn, <Oprnd2>	Rd := Rn - Oprnd2 {CPSR}
	SBC{S} Rd, Rn, <Oprnd2>	Rd := Rn + Oprnd2 + Carry {CPSR}
	RSB{S} Rd, Rn, <Oprnd2>	Rd := Oprnd2 - Rn {CPSR}
	RSC{S} Rd, Rn, <Oprnd2>	Rd := Oprnd2 - Rn - NOTCarry {CPSR}
	MUL{S} Rd, Rm, Rs	Rd := Rm * Rs {CPSR}
	MLA{S} Rd, Rm, Rs, Rn	Rd := Rm * Rs + Rn {CPSR}
Logical	CLZ Rd, Rm	Rd := # leading zero in Rm
	AND{S} Rd, Rn, <Oprnd2>	Rd := Rn AND Oprnd2 {CPSR}
	EOR{S} Rd, Rn, <Oprnd2>	Rd := Rn EXOR Oprnd2 {CPSR}
	ORR{S} Rd, Rn, <Oprnd2>	Rd := Rn OR Oprnd2 {CPSR}
	TST Rn, <Oprnd2>	Update CPSR on Rn AND Oprnd2
	TEQ Rn, <Oprnd2>	Update CPSR on Rn EOR Oprnd2
	BIC{S} Rd, Rn, <Oprnd2>	Rd := Rn AND NOT Oprnd2 {CPSR}
	NOP	R0 := R0
	CMP Rd, <Oprnd2>	Update CPSR on Rn - Oprnd2
	B{cond} label	R15 := label
Branch	BL{cond} label	R14 := R15-4; R15 := label
	SWP Rd, Rm	temp := Rn; Rn := Rm; Rd := temp
Load	LDR Rd, <a_mode2>	Rd := address
	LDM <a_mode4L> Rd{!}, <reglist>	Load list of registers from [Rd]
Store	STR Rd, <a_mode2>	[address] := Rd
	STM <a_mode4S> Rd{!}, <reglist>	Store list of registers to [Rd]
SWI	SWI <immed_24>	Software Interrupt

Addressing Mode 2 - Data Transfer

Pre-indexed	Immediate offset	[Rn, #+/-<immed_12>{!}]
	Zero offset	[Rn]
	Register offset	[Rn, +/-Rm]{!}
	Scaled register offset	[Rn, +/-Rm, LSL #<immed_5>]{!}
		[Rn, +/-Rm, LSR #<immed_5>]{!}
		[Rn, +/-Rm, ASR #<immed_5>]{!}
		[Rn, +/-Rm, ROR #<immed_5>]{!}
		[Rn, +/-Rm, RRRX]{!}
Post-indexed	Immediate offset	[Rn], #+/-<immed_12>
	Register offset	[Rn], +/-Rm
	Zero offset	[Rn]
	Scaled register offset	[Rn], +/-Rm, LSL #<immed_5>
		[Rn], +/-Rm, LSR #<immed_5>
		[Rn], +/-Rm, ASR #<immed_5>
		[Rn], +/-Rm, ROR #<immed_5>
		[Rn], +/-Rm, RRRX

Key to tables

{cond}	See Condition Field
<Oprnd2>	See Operand 2
{S}	Updates CPSR if present
<immed>	Constant
<a_mode2>	See Addressing Mode 2
<a_mode4>	See Addressing Mode 4
<reglist>	List of registers with commas
{!}	Updates base register if present

