컴퓨터 구조

<ALU 시뮬레이터 구현>

제출일 : 2019.10.00

학번 이름

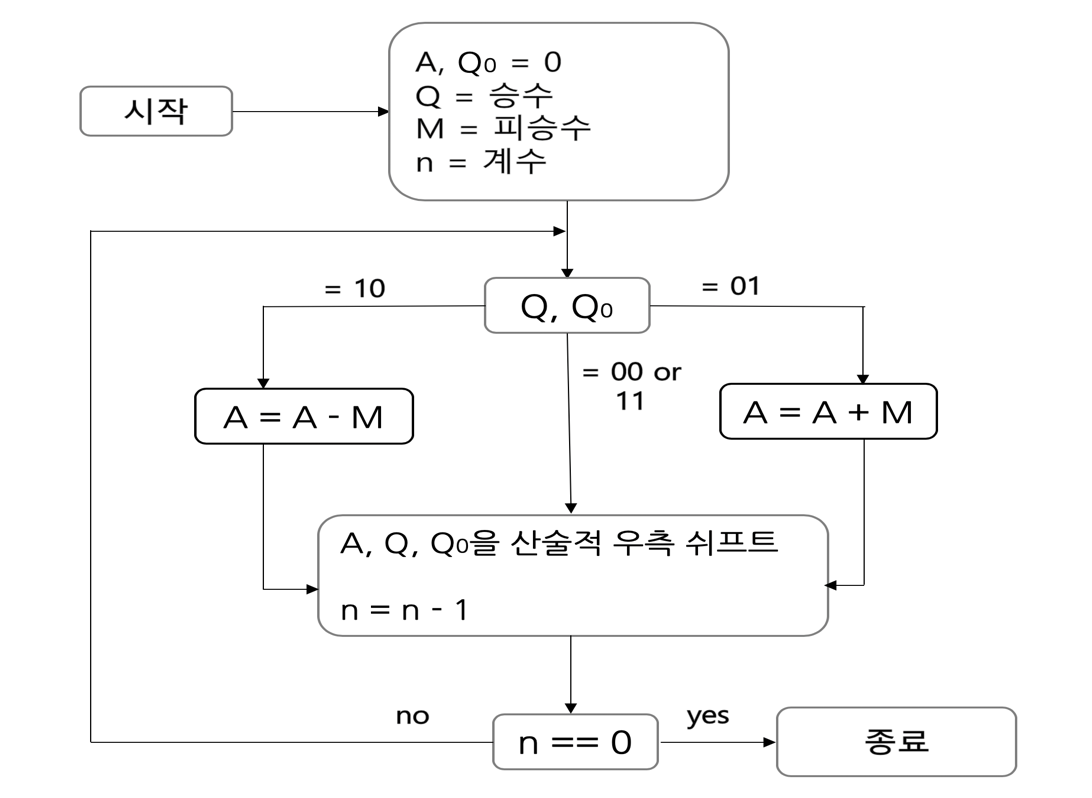
1. 문제 정의

1. Idea 및 알고리즘

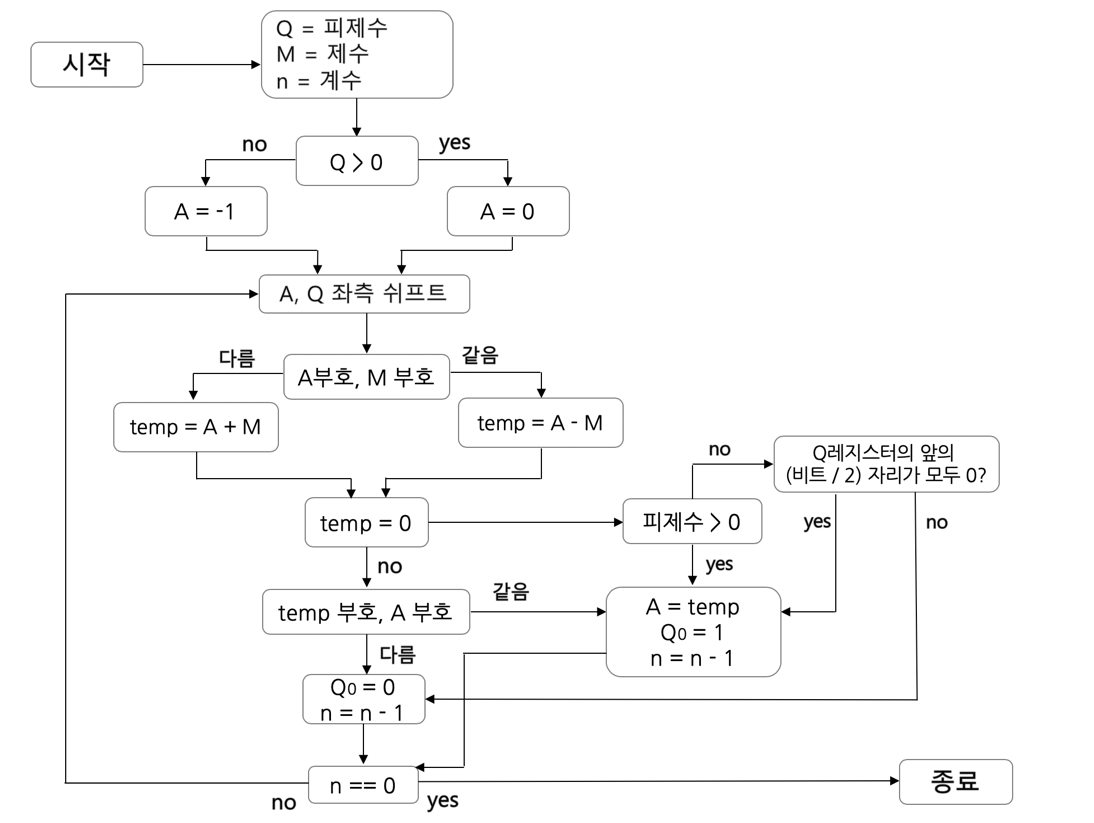
1. 수행 결과

1. 토의 사항
2. 문제 정의

2. Idea 및 알고리즘

2.3곱셈 / Booth Algorithm

2.4나눗셈



3. 수행결과

|  |
| --- |
| 1. 입력 받은 정수: 7 3 |
|  |
| **덧셈 결과**  이진수 : 00000000000000000000000000001010  십진수 : 10 |
| **뺄셈 결과**  이진수 : 00000000000000000000000000000100  십진수 : 4 |
|  |
| **곱셈 결과**  이진수 : 0000000000000000000000000000000000000000000000000000000000010101  십진수 : 21 |
|  |
| **나눗셈 결과**  나머지 : 00000000000000000000000000000001  몫 : 00000000000000000000000000000010  나머지 십진수 : 1  몫 십진수 : 2 |
| 2. 입력 받은 정수: 7 -3 |
|  |
| **덧셈 결과**  이진수 : 00000000000000000000000000000100  십진수 : 4 |
| **뺄셈 결과**  이진수 : 00000000000000000000000000001010  십진수 : 10 |
|  |
| **곱셈 결과**  이진수 : 1111111111111111111111111111111111111111111111111111111111101011  십진수 : -21 |
|  |
| **나눗셈 결과**  나머지 이진수 : 00000000000000000000000000000001  몫 이진수 : 11111111111111111111111111111110  나머지 십진수 : 1  몫 십진수 : -2 |
| 3. 입력 받은 정수: -7 3 |
|  |
| **덧셈 결과**  이진수 : 11111111111111111111111111111100  십진수 : -4 |
| **뺄셈 결과**  이진수 : 11111111111111111111111111110110  십진수 : -10 |
|  |
| **곱셈 결과**  이진수 : 1111111111111111111111111111111111111111111111111111111111101011  십진수 : -21 |
|  |
| **나눗셈 결과**  나머지 이진수 : 11111111111111111111111111111111  몫 이진수 : 11111111111111111111111111111110  나머지 십진수 : -1  몫 십진수 : -2 |
| 4. 입력 받은 정수: -7 -3 |
|  |
| **덧셈 결과**  이진수 : 11111111111111111111111111110110  십진수 : -10 |
| **뺄셈 결과**  이진수 : 11111111111111111111111111111100  십진수 : -4 |
|  |
| **곱셈 결과**  이진수 : 0000000000000000000000000000000000000000000000000000000000010101  십진수 : 21 |
|  |
| **나눗셈 결과**  나머지 이진수 : 11111111111111111111111111111111  몫 이진수 : 00000000000000000000000000000010  나머지 십진수 : -1  몫 십진수 : 2 |
| 5. 입력 받은 정수: -5 2 |
|  |
| **덧셈 결과**  이진수 : 11111111111111111111111111111101  십진수 : -3 |
| **뺄셈 결과**  이진수 : 11111111111111111111111111111001  십진수 : -7 |
|  |
| **곱셈 결과**  이진수 : 1111111111111111111111111111111111111111111111111111111111110110  십진수 : -10 |
|  |
| **나눗셈 결과**  나머지 이진수 : 11111111111111111111111111111111  몫 이진수 : 11111111111111111111111111111110  나머지 십진수 : -1  몫 십진수 : -2 |
| 6. 입력 받은 정수: 600 20 |
|  |
| **덧셈 결과**  이진수 : 00000000000000000000001001101100  십진수 : 620 |
| **뺄셈 결과**  이진수 : 00000000000000000000001001000100  십진수 : 580 |
|  |
| **곱셈 결과**  이진수 : 0000000000000000000000000000000000000000000000000010111011100000  십진수 : 12000 |
|  |
| **나눗셈 결과**  나머지 이진수 : 00000000000000000000000000000000  몫 이진수 : 00000000000000000000000000011110  나머지 십진수 : 0  몫 십진수 : 30 |
| 7. 입력 받은 정수: 1234567 1234 |
|  |
| **덧셈 결과**  이진수 : 00000000000100101101101101011001  십진수 : 1235801 |
| **뺄셈 결과**  이진수 : 00000000000100101101000110110101  십진수 : 1233333 |
|  |
| **곱셈 결과**  이진수 : 0000000000000000000000000000000001011010110011100001011010111110  십진수 : 1523455678 |
|  |
| **나눗셈 결과**  나머지 이진수 : 00000000000000000000001000110111  몫 이진수 : 00000000000000000000001111101000  나머지 십진수 : 567  몫 십진수 : 1000 |
| 8. 입력 받은 정수: 2147483647(32비트 최대 정수) 2 |
|  |
| **덧셈 결과**  이진수 : 오버플로우 (2147483647 = 32비트 최대수이기 때문에 오버플로우가 발생한다.)  십진수 : 오버플로우 |
| **뺄셈 결과**  이진수 : 01111111111111111111111111111101  십진수 : 2147483645 |
|  |
| **곱셈 결과**  이진수 : 0000000000000000000000000000000011111111111111111111111111111110  십진수 : 4294967294 |
|  |
| **나눗셈 결과**  나머지 이진수 : 00000000000000000000000000000001  몫 이진수 : 00111111111111111111111111111111  나머지 십진수 : 1  몫 십진수 : 1073741823 |
| 9. 입력 받은 정수: -2147483648 2 (32비트 최소 정수) |
|  |
| **덧셈 결과**  이진수 : 10000000000000000000000000000010  십진수 : -2147483646 |
| **뺄셈 결과**  이진수 : 언더플로우 (-2147483648 이 32비트의 최소 정수이기 때문에 언더플로우가 발생한다.)  십진수 : 언더플로우 |
|  |
| **곱셈 결과**  이진수 : 연산오류  십진수 : 연산오류  \*\* 연산 과정에서 flag\_v가 1이 되기 때문에 연산이 정상적으로 작동하지 않는다. |
|  |
| **나눗셈 결과**  나머지 이진수 : 00000000000000000000000000000000  몫 이진수 : 11000000000000000000000000000000  나머지 십진수 : 0  몫 십진수 : -1073741824 |
| 10. 입력 받은 정수: -0 -2147483648 (32비트 최소 정수) |
|  |
| **덧셈 결과**  이진수 : 10000000000000000000000000000000  십진수 : -2147483648 |
| **뺄셈 결과**  이진수 : 10000000000000000000000000000000  십진수 : -2147483648 |
|  |
| **곱셈 결과**  이진수 : 000000000000000000000000000000000000000000000000000000000000000  십진수 : 0 |
|  |
| **나눗셈 결과**  이진수 : 연산오류  십진수 : 연산오류  \*\* 연산 과정에서 flag\_v가 1이 되기 때문에 연산이 정상적으로 작동하지 않는다. |
| 11. 입력 받은 정수: 2147483647 (32비트 최대 정수) 0 |
|  |
| **덧셈 결과**  이진수 : 01111111111111111111111111111111  십진수 : 2147483647 |
| **뺄셈 결과**  이진수 : 01111111111111111111111111111111  십진수 : 2147483647 |
|  |
| **곱셈 결과**  이진수 : 000000000000000000000000000000000000000000000000000000000000000  십진수 : 0 |
|  |
| **나눗셈 결과**  \*\* 분모가 0이 될 수는 없으므로 나눌 수 없다. |
| 12. 입력한 정수 : 2147483648 3 / 입력한 정수 : -2147483649 3 |
|  |
| 32비트가 표현 가능한 수의 범위를 벗어났으므로 가능한 범위를 벗어났다고 출력한다 |

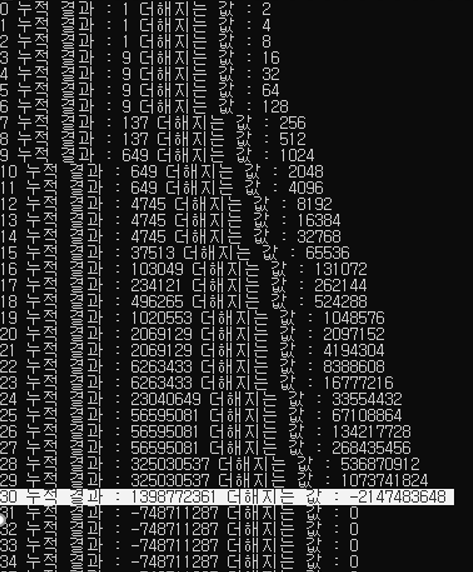
5. 토의 사항

1. 나눗셈 알고리즘에서 교수님께서 말씀해주신 내용과 강의 자료를 참고하여 알고리즘을 작성하였으나, Q / M을 할 때 Q가 음수이며, 딱 나눠 떨어지는 경우에 대하여 올바른 결과를 못하였다. 놓친 부분이 있다고 생각하여 인터넷에서도 도움을 얻으려 하였으나 원하는 정보를 얻지 못하여 각 step마다 손으로 직접 써가며 프로그램 결과와 비교하여 놓치는 부분을 찾으려고 노력하였다. 많은 가설들을 세워가며 결국 Q(피제수)가 음수이며 A = A – M 또는 A = A + M의 결과가 0일때 Q의 레지스터 상태에 따라 연산의 성공/실패가 달라지는 것을 깨달았다. 언급한 상황에서 Q레지스터의 앞 (bit/2)자리가 모두 0이면 성공으로 처리하고 아니면 실패로 처리하고 나서야 올바른 결과를 얻을 수 있었다.

2. 곱셈 연산시 32비트로 입력 받지만 결과 값은 A레지스터와 Q레지스터를 모두 사용해서 64비트로 나온다. 처음 2진수에서 10진수로 변환하는 함수 **(BinaryToDecimal)**를 작성할 때는 32비트까지 지원하는 int형 변수를 사용하여 올바르지 않은 결과 값이 나왔지만, 이를 64비트까지 지원하는 \_\_int64 변수로 수정하여 문제를 해결했다.

(BinaryToDecimal 함수에서 int 형 변수 사용시 결과 값)





\* 누적 결과 : 이진수 최하위 비트부터 2n값 더한 누적 값

\* 더해지는 값 : 2n

(BinaryToDecimal 함수에서 \_\_int64 형 변수 사용시 결과 값)



3. 병렬 가산기에서 flag\_v가 뜻하는 것은 오버플로우의 발생 여부이다. 내부 캐리와 외부 캐리의 XOR연산을 이용하여 v가 1이 되었을 때 오버플로우가 발생했다는 것을 알 수 있는데, 이는 덧셈이나 뺄셈 외에도 덧셈, 뺄셈 연산을 반복하여 사용하는 곱셈, 나눗셈 연산에서도 오버플로우 발생 여부를 확인 하기 위해 사용할 수 있다. 곱셈, 나눗셈의 경우에는 이를 연속해서 사용하기 때문에 중간에 오버플로우 혹은 언더플로우가 발생하게 되면 마지막 결과에 ‘오버플로우/언더플로우로 인한 연산 오류’ 라는 문구를 출력하였다.

입력한 정수 : 0 -2147483648  (결과)