

# Lookup Tables

## Objective

To discuss lookup tables and how to use them to sacrifice storage space to increase computation time.

## What Are Lookup Tables

**Lookup tables** are static arrays that sacrifices memory storage in place of a simple array index lookup of precalculated values. In some examples, a lookup table is not meant to speed a process, but simply an elegant solution to a problem.

Lets look at some examples to see why these are useful.

## Why Use Lookup Tables

### Simple Example: Convert Potentiometer Voltage to Angle

Lets make some assumptions about the system first:

1. Using an 8-bit ADC
2. Potentiometer is linear
3. Potentiometer sweep angle is 180 or 270 degrees
4. Potentiometer all the way left is 0 deg and 0V
5. Potentiometer all the way right (180/270 deg) is ADC Reference Voltage
6. Using a processor that does NOT have a FPU (**F**loating **P**oint arithmetic **U**nit) like the Arm Cortex M3 we use in the LPC1756.

```
double potADCToDegrees(uint8_t adc)
{
    return ((double)(adc))*(270/256);
}
```

*Code Block 1. Without Lookup*

```
const double potentiometer_angles[256] =
{
//  [ADC] = Angle
[0] = 0.0,
[1] = 1.0546875,
[2] = 2.109375,
[3] = 3.1640625,
[4] = 4.21875,
[5] = 5.2734375,
[6] = 6.328125,
[7] = 7.3828125,
[8] = 8.4375,
[9] = 9.4921875,
[10] = 10.546875,
[11] = 11.6015625,
[12] = 12.65625,
[13] = 13.7109375,
[14] = 14.765625,
[15] = 15.8203125,
[16] = 16.875,
[17] = 17.9296875,
[18] = 18.984375,
[19] = 20.0390625,
[20] = 21.09375,
[21] = 22.1484375,
[22] = 23.203125,
[23] = 24.2578125,
[24] = 25.3125,
[25] = 26.3671875,
[26] = 27.421875,
[27] = 28.4765625,
[28] = 29.53125,
[29] = 30.5859375,
[30] = 31.640625,
[31] = 32.6953125,
[32] = 33.75,
```

[33] = 34.8046875,  
[34] = 35.859375,  
[35] = 36.9140625,  
[36] = 37.96875,  
[37] = 39.0234375,  
[38] = 40.078125,  
[39] = 41.1328125,  
[40] = 42.1875,  
[41] = 43.2421875,  
[42] = 44.296875,  
[43] = 45.3515625,  
[44] = 46.40625,  
[45] = 47.4609375,  
[46] = 48.515625,  
[47] = 49.5703125,  
[48] = 50.625,  
[49] = 51.6796875,  
[50] = 52.734375,  
[51] = 53.7890625,  
[52] = 54.84375,  
[53] = 55.8984375,  
[54] = 56.953125,  
[55] = 58.0078125,  
[56] = 59.0625,  
[57] = 60.1171875,  
[58] = 61.171875,  
[59] = 62.2265625,  
[60] = 63.28125,  
[61] = 64.3359375,  
[62] = 65.390625,  
[63] = 66.4453125,  
[64] = 67.5,  
[65] = 68.5546875,  
[66] = 69.609375,  
[67] = 70.6640625,  
[68] = 71.71875,

```
□[69] □= 72.7734375,  
□[70] □= 73.828125,  
□[71] □= 74.8828125,  
□[72] □= 75.9375,  
□[73] □= 76.9921875,  
□[74] □= 78.046875,  
□[75] □= 79.1015625,  
□[76] □= 80.15625,  
□[77] □= 81.2109375,  
□[78] □= 82.265625,  
□[79] □= 83.3203125,  
□[80] □= 84.375,  
□[81] □= 85.4296875,  
□[82] □= 86.484375,  
□[83] □= 87.5390625,  
□[84] □= 88.59375,  
□[85] □= 89.6484375,  
□[86] □= 90.703125,  
□[87] □= 91.7578125,  
□[88] □= 92.8125,  
□[89] □= 93.8671875,  
□[90] □= 94.921875,  
□[91] □= 95.9765625,  
□[92] □= 97.03125,  
□[93] □= 98.0859375,  
□[94] □= 99.140625,  
□[95] □= 100.1953125,  
□[96] □= 101.25,  
□[97] □= 102.3046875,  
□[98] □= 103.359375,  
□[99] □= 104.4140625,  
□[100] □= 105.46875,  
    // ...  
□[240] □= 253.125,  
□[241] □= 254.1796875,  
□[242] □= 255.234375,
```

```

    [243] = 256.2890625,
    [244] = 257.34375,
    [245] = 258.3984375,
    [246] = 259.453125,
    [247] = 260.5078125,
    [248] = 261.5625,
    [249] = 262.6171875,
    [250] = 263.671875,
    [251] = 264.7265625,
    [252] = 265.78125,
    [253] = 266.8359375,
    [254] = 267.890625,
    [255] = 268.9453125,
    [256] = 270
};

inline double potADCToDegrees(uint8_t adc)
{
    return potentiometer_angles[adc]}

```

### Code Block 2. With Lookup

With the two examples, it may seem trivial since the *WITHOUT* case is only "really" doing one calculation, multiplying the **uint8\_t** with (270/256) since the compiler will most likely optimize this value to its result. But if you take a look at the assembly, the results may shock you.

## Look up Table Disassembly

```

00016e08 <main>:
main():
/var/www/html/SJSU-Dev/firmware/Experiements/L5_Application/main.cpp:322
    [254] = 268.9411765,
    [255] = 270
};
int main(void)
{
    16e08:0b082    sub sp, #8
/var/www/html/SJSU-Dev/firmware/Experiements/L5_Application/main.cpp:323
    volatile double a = potentiometer_angles[15];

```

```

16e0a:  a303      add r3, pc, #12; (adr r3, 16e18 <main+0x10>)
16e0c:  e9d3 2300  ldrd r2, r3, [r3]
16e10:  e9cd 2300  strd r2, r3, [sp]
16e14:  e7fe      b.n 16e14 <main+0xc>
16e16:  bf00      nop
16e18:  c3b9a8ae  .word 0xc3b9a8ae 16e1c:  402fc3c3  .word 0x402fc3c3

```

### Code Block 3. Dissassembly of Look up Table

Looks about right. You can see at **16e0a** the software is retrieving data from the lookup table, and then it is loading it into the double which is on the stack.

## Double Floating Point Disassembly

```

00017c64 <__adddf3>:
__aeabi_dadd():
17c64:  b530      push {r4, r5, lr}
17c66:  ea4f 0441  mov.w r4, r1, lsl #1
17c6a:  ea4f 0543  mov.w r5, r3, lsl #1
17c6e:  ea94 0f05  teq r4, r5
17c72:  bf08      it eq
17c74:  ea90 0f02  teqeq r0, r2
17c78:  bf1f      itttt ne
17c7a:  ea54 0c00  orrsne.w ip, r4, r0
17c7e:  ea55 0c02  orrsne.w ip, r5, r2
17c82:  ea7f 5c64  mvnsne.w ip, r4, asr #21
17c86:  ea7f 5c65  mvnsne.w ip, r5, asr #21
17c8a:  f000 80e2  beq.w 17e52 <__adddf3+0x1ee>
17c8e:  ea4f 5454  mov.w r4, r4, lsr #21
17c92:  ebd4 5555  rsbs r5, r4, r5, lsr #21
17c96:  bfb8      it lt
17c98:  426d      neglt r5, r5
17c9a:  dd0c      ble.n 17cb6 <__adddf3+0x52>
17c9c:  442c      add r4, r5
17c9e:  ea80 0202  eor.w r2, r0, r2
17ca2:  ea81 0303  eor.w r3, r1, r3
17ca6:  ea82 0000  eor.w r0, r2, r0

```

```

17caa:  ea83 0101  eor.w  r1, r3, r1
17cae:  ea80 0202  eor.w  r2, r0, r2
17cb2:  ea81 0303  eor.w  r3, r1, r3
17cb6:  2d36      cmp    r5, #54; 0x36
17cb8:  bf88      it     hi
17cba:  bd30      pophi  {r4, r5, pc}
17cbc:  f011 4f00  tst.w  r1, #2147483648; 0x80000000
17cc0:  ea4f 3101  mov.w  r1, r1, lsl #12
17cc4:  f44f 1c80  mov.w  ip, #1048576; 0x100000
17cc8:  ea4c 3111  orr.w  r1, ip, r1, lsr #12
17ccc:  d002      beq.n  17cd4 <__adddf3+0x70>
17cce:  4240      negs   r0, r0
17cd0:  eb61 0141  sbc.w  r1, r1, r1, lsl #1
17cd4:  f013 4f00  tst.w  r3, #2147483648; 0x80000000
17cd8:  ea4f 3303  mov.w  r3, r3, lsl #12
17cdc:  ea4c 3313  orr.w  r3, ip, r3, lsr #12
17ce0:  d002      beq.n  17ce8 <__adddf3+0x84>
17ce2:  4252      negs   r2, r2
17ce4:  eb63 0343  sbc.w  r3, r3, r3, lsl #1
17ce8:  ea94 0f05  teq    r4, r5
17cec:  f000 80a7  beq.w  17e3e <__adddf3+0x1da>
17cf0:  f1a4 0401  sub.w  r4, r4, #1
17cf4:  f1d5 0e20  rsbs   lr, r5, #32
17cf8:  db0d      blt.n  17d16 <__adddf3+0xb2>
17cfa:  fa02 fc0e  lsl.w  ip, r2, lr
17cfe:  fa22 f205  lsr.w  r2, r2, r5
17d02:  1880      adds   r0, r0, r2
17d04:  f141 0100  adc.w  r1, r1, #0
17d08:  fa03 f20e  lsl.w  r2, r3, lr
17d0c:  1880      adds   r0, r0, r2
17d0e:  fa43 f305  asr.w  r3, r3, r5
17d12:  4159      adcs   r1, r3
17d14:  e00e      b.n    17d34 <__adddf3+0xd0>
17d16:  f1a5 0520  sub.w  r5, r5, #32
17d1a:  f10e 0e20  add.w  lr, lr, #32
17d1e:  2a01      cmp    r2, #1

```

```

17d20:  fa03 fc0e  lsl.w ip, r3, lr
17d24:  bf28      itcs
17d26:  f04c 0c02  orrcs.w ip, ip, #2
17d2a:  fa43 f305  asr.w r3, r3, r5
17d2e:  18c0      adds r0, r0, r3
17d30:  eb51 71e3  adcs.w r1, r1, r3, asr #31
17d34:  f001 4500  and.w r5, r1, #2147483648; 0x80000000
17d38:  d507      bpl.n 17d4a <__adddf3+0xe6>
17d3a:  f04f 0e00  mov.w lr, #0
17d3e:  f1dc 0c00  rsbs ip, ip, #0
17d42:  eb7e 0000  sbcs.w r0, lr, r0
17d46:  eb6e 0101  sbc.w r1, lr, r1
17d4a:  f5b1 1f80  cmp.w r1, #1048576; 0x100000
17d4e:  d31b      bcc.n 17d88 <__adddf3+0x124>
17d50:  f5b1 1f00  cmp.w r1, #2097152; 0x200000
17d54:  d30c      bcc.n 17d70 <__adddf3+0x10c>
17d56:  0849      lsrs r1, r1, #1
17d58:  ea5f 0030  movs.w r0, r0, rrx
17d5c:  ea4f 0c3c  mov.w ip, ip, rrx
17d60:  f104 0401  add.w r4, r4, #1
17d64:  ea4f 5244  mov.w r2, r4, lsl #21
17d68:  f512 0f80  cmn.w r2, #4194304; 0x400000
17d6c:  f080 809a  bcs.w 17ea4 <__adddf3+0x240>
17d70:  f1bc 4f00  cmp.w ip, #2147483648; 0x80000000
17d74:  bf08      iteq
17d76:  ea5f 0c50  movseq.w ip, r0, lsr #1
17d7a:  f150 0000  adcs.w r0, r0, #0
17d7e:  eb41 5104  adc.w r1, r1, r4, lsl #20
17d82:  ea41 0105  orr.w r1, r1, r5
17d86:  bd30      pop {r4, r5, pc}
17d88:  ea5f 0c4c  movs.w ip, ip, lsl #1
17d8c:  4140      adcs r0, r0
17d8e:  eb41 0101  adc.w r1, r1, r1
17d92:  f411 1f80  tst.w r1, #1048576; 0x100000
17d96:  f1a4 0401  sub.w r4, r4, #1
17d9a:  d1e9      bne.n 17d70 <__adddf3+0x10c>

```



```

17d9c: 0f01 0f00  iteq r1, #0
17da0: bf04      itt eq
17da2: 4601      moveq r1, r0
17da4: 2000      moveq r0, #0
17da6: fab1 f381 clz r3, r1
17daa: bf08      it eq
17dac: 3320      addeq r3, #32
17dae: f1a3 030b sub.w r3, r3, #11
17db2: f1b3 0220 subs.w r2, r3, #32
17db6: da0c      bge.n 17dd2 <__adddf3+0x16e>
17db8: 320c      adds r2, #12
17dba: dd08      ble.n 17dce <__adddf3+0x16a>
17dbc: f102 0c14 add.w ip, r2, #20
17dc0: f1c2 020c rsb r2, r2, #12
17dc4: fa01 f00c lsl.w r0, r1, ip
17dc8: fa21 f102 lsr.w r1, r1, r2
17dcc: e00c      b.n 17de8 <__adddf3+0x184>
17dce: f102 0214 add.w r2, r2, #20
17dd2: bfd8      it le
17dd4: f1c2 0c20 rsble ip, r2, #32
17dd8: fa01 f102 lsl.w r1, r1, r2
17ddc: fa20 fc0c lsr.w ip, r0, ip
17de0: bfdc      itt le
17de2: ea41 010c orrle.w r1, r1, ip
17de6: 4090      lsllle r0, r2
17de8: 1ae4      subs r4, r4, r3
17dea: bfa2      ittt ge
17dec: eb01 5104 addge.w r1, r1, r4, lsl #20
17df0: 4329      orrge r1, r5
17df2: bd30      popge {r4, r5, pc}
17df4: ea6f 0404 mvn.w r4, r4
17df8: 3c1f      subs r4, #31
17dfa: da1c      bge.n 17e36 <__adddf3+0x1d2>
17dfc: 340c      adds r4, #12
17dfe: dc0e      bgt.n 17e1e <__adddf3+0x1ba>
17e00: f104 0414 add.w r4, r4, #20

```

```

17e04: f1c4 0220 rsb r2, r4, #32
17e08: fa20 f004 lsr.w r0, r4
17e0c: fa01 f302 lsl.w r3, r1, r2
17e10: ea40 0003 orr.w r0, r0, r3
17e14: fa21 f304 lsr.w r3, r1, r4
17e18: ea45 0103 orr.w r1, r5, r3
17e1c: bd30      pop {r4, r5, pc}
17e1e: f1c4 040c rsb r4, r4, #12
17e22: f1c4 0220 rsb r2, r4, #32
17e26: fa20 f002 lsr.w r0, r0, r2
17e2a: fa01 f304 lsl.w r3, r1, r4
17e2e: ea40 0003 orr.w r0, r0, r3
17e32: 4629      mov r1, r5
17e34: bd30      pop {r4, r5, pc}
17e36: fa21 f004 lsr.w r0, r1, r4
17e3a: 4629      mov r1, r5
17e3c: bd30      pop {r4, r5, pc}
17e3e: f094 0f00 teq r4, #0
17e42: f483 1380 eor.w r3, r3, #1048576; 0x100000
17e46: bf06      itte eq
17e48: f481 1180 eoreq.w r1, r1, #1048576; 0x100000
17e4c: 3401      addeq r4, #1
17e4e: 3d01      subne r5, #1
17e50: e74e      b.n 17cf0 <__adddf3+0x8c>
17e52: ea7f 5c64 mvns.w ip, r4, asr #21
17e56: bf18      it ne
17e58: ea7f 5c65 mvnsne.w ip, r5, asr #21
17e5c: d029      beq.n 17eb2 <__adddf3+0x24e>
17e5e: ea94 0f05 teq r4, r5
17e62: bf08      it eq
17e64: ea90 0f02 teqeq r0, r2
17e68: d005      beq.n 17e76 <__adddf3+0x212>
17e6a: ea54 0c00 orrs.w ip, r4, r0
17e6e: bf04      itt eq
17e70: 4619      moveq r1, r3
17e72: 4610      moveq r0, r2

```

```

17e74: 0bd30      pop  {r4, r5, pc}
17e76: 0ea91 0f03  teq  r1, r3
17e7a: 0bf1e      ittt  ne
17e7c: 02100      movne r1, #0
17e7e: 02000      movne r0, #0
17e80: 0bd30      popne {r4, r5, pc}
17e82: 0ea5f 5c54  movs.w ip, r4, lsr #21
17e86: 0d105      bne.n 17e94 <__adddf3+0x230>
17e88: 00040      lsls  r0, r0, #1
17e8a: 04149      adcs  r1, r1
17e8c: 0bf28      it  cs
17e8e: 0f041 4100  orrcs.w r1, r1, #2147483648; 0x80000000
17e92: 0bd30      pop  {r4, r5, pc}
17e94: 0f514 0480  adds.w r4, r4, #4194304; 0x400000
17e98: 0bf3c      itt  cc
17e9a: 0f501 1180  addcc.w r1, r1, #1048576; 0x100000
17e9e: 0bd30      popcc {r4, r5, pc}
17ea0: 0f001 4500  and.w r5, r1, #2147483648; 0x80000000
17ea4: 0f045 41fe  orr.w r1, r5, #2130706432; 0x7f000000
17ea8: 0f441 0170  orr.w r1, r1, #15728640; 0xf00000
17eac: 0f04f 0000  mov.w r0, #0
17eb0: 0bd30      pop  {r4, r5, pc}
17eb2: 0ea7f 5c64  mvns.w ip, r4, asr #21
17eb6: 0bf1a      itte  ne
17eb8: 04619      movne r1, r3
17eba: 04610      movne r0, r2
17ebc: 0ea7f 5c65  mvnseq.w ip, r5, asr #21
17ec0: 0bf1c      itt  ne
17ec2: 0460b      movne r3, r1
17ec4: 04602      movne r2, r0
17ec6: 0ea50 3401  orrs.w r4, r0, r1, lsl #12
17eca: 0bf06      itte  eq
17ecc: 0ea52 3503  orrseq.w r5, r2, r3, lsl #12
17ed0: 0ea91 0f03  teqeq r1, r3
17ed4: 0f441 2100  orrne.w r1, r1, #524288; 0x80000
17ed8: 0bd30      pop  {r4, r5, pc}

```

```
17eda:  bf00      nop
```

#### Code Block 4. Arm Software Floating Point Addition Implementation

This isn't even the full code. This is a function that our calculation function has to run each time it wants to add two doubles together. Also, note that it is not just a straight shot of 202 instructions, because you can see that there are loops in the code where ever you see an instruction's mnemonic that starts with the letter **b** (stands for branch).

## Other Use Cases

- Correlate degrees to radians (assuming degrees are whole numbers)
- Table of cosine or sine given radians or degrees
  - In the radians case, you will need to create your own trivial hashing function to convert radians to an index
- Finding a number of bits SET in a 32-bit number
  - Without a lookup table time complexity is  $O(n)$  where ( $n = 32$ ), the number of bits you want to look through
  - With a lookup table, the time complexity is  $O(1)$ , constant time, and only needs the followin operations
    - 3 bitwise left shifts operations
    - 4 bitwise ANDS operations
    - 4 load from memory addresses
    - 4 binary ADD operations
    - Total of 15 operations total

```
/* Found this on wikipedia! */
/* Pseudocode of the lookup table 'uint32_t bits_set[256]' */
/*
    0b00, 0b01, 0b10, 0b11, 0b100, 0b101, ... */
int bits_set[256] = {    0,    1,    1,    2,    1,    2, // 200+ more entries
/* (this code assumes that 'int' is an unsigned 32-bits wide integer) */
int count_ones(unsigned int x) {
    return bits_set[ x          & 255] + bits_set[(x >>  8) & 255]
        + bits_set[(x >> 16) & 255] + bits_set[(x >> 24) & 255]; }
```

#### Code Block 5. Bits set in a 32-bit number (Found this on wikipedia (look up tables))

There are far more use cases then this, but these are a few.

## Lookup Table Decision Tree

Lookup tables can be used as elegant ways to structure information. In this case, they may not provide a speed up but they will associate indexes with something greater, making your code more readable and easier to maintain. In this example, we will be looking at a matrix of function pointers.

## Example: Replace Decision Tree

See the function below:

```
void makeADecisionRobot(bool power_system_nominal, bool no_obstacles_ahead)
{
    if(power_system_nominal && no_obstacles_ahead) {
        moveForward();
    }
    else if(power_system_nominal && !no_obstacles_ahead) {
        moveOutOfTheWay();
    }
    else if(!power_system_nominal && no_obstacles_ahead) {
        slowDown();
    }
    else {
        emergencyStop();
    }
}
```

Code Block 6. Typical Decision Tree

```
void (* decision_matrix[2][2])(void) =
{
    [1][1] = moveForward,
    [1][0] = moveOutOfTheWay,
    [0][1] = slowDown,
    [0][0] = emergencyStop,
};

void makeADecisionRobot(bool power_system_nominal, bool no_obstacles_ahead)
{
    decision_matrix[power_system_nominal][no_obstacles_ahead]();
}
```

Code Block 7. Lookup Table Decision Tree

The interesting thing about the decision tree is that it is also more optimal in that, it takes a few instructions to do the look up from memory, then the address of the procedure [function] is looked up and executed, where the former required multiple read instructions and comparison instructions.

This pattern of lookup table will be most useful to us for the interrupts lab assignment.

---

Revision #11

Created 7 years ago by [Admin](#)

Updated 5 years ago by [sree harsha](#)

?