UNIT TEST
M68K pour okba

Interpréter Instruction:

ABCD — Add Decimal with Extend (M68000 Family) — ABCD

Operation: Source10 + Destination10 + X → Destination

Assembler Syntax: ABCD Dy,Dx
ABCD – (Ay), – (Ax)

Attributes: Size = (Byte)

Description: Adds the source operand to the destination operand along with the extend bit, and stores the result in the destination location. The addition is performed using binary-coded decimal arithmetic. The operands, which are packed binary-coded decimal numbers, can be addressed in two different ways:

1. Data Register to Data Register: The operands are contained in the data registers specified in the instruction.
2. Memory to Memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

This operation is a byte operation only.

Condition Codes:

X	N	Z	V	C
•	U	•	U	•

X — Set the same as the carry bit.
N — Undefined.
Z — Cleared if the result is nonzero; unchanged otherwise.
V — Undefined.
C — Set if a decimal carry was generated; cleared otherwise.

NOTE

Normally, the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	REGISTER Rx	1	0	0	0	0	R/M	REGISTER Ry

Instruction Fields:

Register Rx field—Specifies the destination register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

R/M field—Specifies the operand addressing mode.
0 — The operation is data register to data register.
1 — The operation is memory to memory.

Register Ry field—Specifies the source register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

Ici je vais expliquer comment décortiquer une des instructions les plus simples : ABCD

L'opération :

Ici rien de complexe, on voit que l'instruction fait une addition comme ceci « Source + destination + X »

La source, est le registre le plus proche de du bit 0 dans la structure binaire de l'opcode . Ici Ry

La destination et le registre qui suit. Ici Rx

X est tout simplement le Flag X

La taille :

Simplement les tailles possibles des datas utiliser par l'opcode. Donc la taille de source et destination.

Ici seulement le BYTE est autorisé, donc les valeurs peuvent aller entre 0x0 et 0xFF.

La description :

Ici il suffit juste de bien comprendre, c'est souvent un résumé de ce qu'il y a au dessus avec en plus des informations importantes, une description détaillée en clair.

Donc, celle-ci nous réexplique l'opération mais en nous indiquant ou doit être stocké le resultat. On nous dit que le resultat doit être stocké dans la destination.

Notre instruction execute donc :

« destination = source + destination + X »

Ensuite, information très importante, l'opération est faite en utilisant le BCD(Binary Coded Decimal).

Ça peut paraître compliqué, mais en fait non, juste au lieu d'utiliser l'hexadecimal on utilise le decimal. Je m'explique.

En Hex, on a une range de 0x0 à 0xF.
En BCD, tout simplement, on a une range de 0x0 à 0x9.

Donc par exemple « 0x01 + 0x03 = 0x04 », facile.
En revanche, en BCD « 0x09 + 0x01 = 0x10» alors qu'en hexadecimal « 0x9 + 0x1 = 0xA ».

Ensuite, on nous explique que l'instruction à deux mode.

Un mode DataRegister to DataRegister.
Dans ce cas la, la source et la destination sont des DataRegisters du M68K.

Un mode Memory To Memory.
Dans ce cas la, l'instruction va utiliser les AddressRegister du CPU et appliquer un calcul pour trouver l'EA(Effective Address).
Ici le calcul est précisé, le mode est EA_ADDRESS_PRE_DECREMENT.
(Attention, ici le mode est précisé, mais dans d'autres cas, tout les modes sont possibles, alors il faut trouver le mode en regardant la structure binaire de l'opcode).

Les FLAGS :

Ici tout est bien décrit.
Je te laisse regarder la doc ici pour comprendre comment fonctionne chaque flag.
http://mrjester.hapisan.com/04_MC68/Sect06Part01/Index.html

Comme tu peux le voir, cette partie explique comment sont traité les flags.

Certains ne sont pas affecté. D'autres oui. Et d'autres avec certaines conditions. Comme par exemple le flag Z, normalement si le resultat est égal à 0, il doit être set, mais la on nous indique que sont état reste inchangé dans ce cas là. Ou encore le flag X, qui est une copie du flag C dans cette instruction.

La structure binaire :

Simple, juste comment structurer son opcode pour faire ce que l'on veut.

Si on set RM, alors on est en mode Memory to Memory,
sinon, DataRegister to DataRegister.

Pour Rx, on doit juste choisir le numero du registre.
Si par exemple on met 010(2 en decimal), on utilisera le registre 2.

Pareil pour Ry.

Maintenant combinont tout ça.

Imaginons que je veux que mon opcode execute cette operation :
« DataRegister[6] = DataRegister[4] + DataRegister[6] + X »

Il faut Reset RM(lui donner une valeur de 0),
mettre Rx(la destination) à 110(6 en decimal),
mettre Ry(la source) à 100(4 en decimal).

Avec ça, on peu créer l'opcode :
1100 Rx(110) 10000 RM(0) Ry(100)

La structure binaire de l'opcode est donc :
1100110100000100

Ce qui nous donnes en Hex :
0xCD04

Et voilà ! On a notre opcode :D

Maintenant passons au chose sérieuse.
Dans ce qui suit je vais expliquer comment créer une fonction de test pour un opcode.

Je vais expliquer comment trouver ce qu'il faut tester,
ce que je dois recevoir en Output, et comment programmer un test.

Ce que tu dois tester :

Tester l'operation avec ces modes la :

Dn (DataRegister)

EA Mode :
An (AddressRegister)

Tester les flags

L'output :

Ici tu as le choix sois tu print dans la console
Sois tu écris dans un fichier.

La forme de l'input doit être comme ceci :

```
Start Test_ABCD()
        M68k :: Launch OpcodeABCD
                Test BCD Operation with X Reset Passed
        M68k :: Launch OpcodeABCD
                Test BCD Operation with X Set Passed
        M68k :: Launch OpcodeABCD
                Test C_FLAG and X_FLAG Passed
        M68k :: Launch OpcodeABCD
                Test Z_FLAG Passed
End Test_ABCD()
```

Les lignes avec le « M68K » sont automatiquement généré par le CPU, en revanche les autres lignes doivent être écrite dans le test.

Exemple : UnitTest complet pour ABCD :

```
void Test_ABCD()
{
    std::cout << "Start Test_ABCD()" << std::endl;

    CPU_STATE_DEBUG state;

    //Test BCD Operation with X Reset
    state.CCR = 0x0000;
    state.registerData[4] = 0x4;
    state.registerData[6] = 0x9;

    M68k::SetCpuState(state);

    M68k::ExecuteOpcode(0xCD04);

    state = M68k::GetCpuState();

    if(state.registerData[6] == 0x13)
    {
        std::cout << "\t\tTest BCD Operation with X Reset Passed" << std::endl;
    }
    else
    {
        std::cout << "\t\tTest BCD Operation with X Reset Failed" << std::endl;
    }

    //Test BCD Operation with X Set
    state.CCR = 0x0000;
    BitSet(state.CCR, X_FLAG);
    state.registerData[4] = 0x4;
    state.registerData[6] = 0x9;

    M68k::SetCpuState(state);

    M68k::ExecuteOpcode(0xCD04);

    state = M68k::GetCpuState();

    if(state.registerData[6] == 0x14)
    {
        std::cout << "\t\tTest BCD Operation with X Set Passed" << std::endl;
    }
    else
    {
        std::cout << "\t\tTest BCD Operation with X Set Failed" << std::endl;
    }
```

```
    //Test C_FLAG and X_FLAG
    state.CCR = 0x0000;
    state.registerData[4] = 0x99;
    state.registerData[6] = 0x1;

    M68k::SetCpuState(state);

    M68k::ExecuteOpcode(0xCD04);

    state = M68k::GetCpuState();

    //et on test
    if(TestFlag(state.CCR, 1, 0, 0, 0, 1))
    {
        std::cout << "\t\tTest C_FLAG and X_FLAG Passed" << std::endl;
    }
    else
    {
        std::cout << "\t\tTest C_FLAG and X_FLAG Failed" << std::endl;
    }

    //Test Z_FLAG
    state.CCR = 0x0000;
    BitSet(state.CCR, Z_FLAG);
    state.registerData[4] = 0x2;
    state.registerData[6] = 0x1;

    M68k::SetCpuState(state);

    M68k::ExecuteOpcode(0xCD04);

    state = M68k::GetCpuState();

    //et on test
    if(TestFlag(state.CCR, 0, 0, 0, 0, 0))
    {
        std::cout << "\t\tTest Z_FLAG Passed" << std::endl;
    }
    else
    {
        std::cout << "\t\tTest Z_FLAG Failed" << std::endl;
    }

    //indique la fin du test
    std::cout << "End Test_ABCD()" << std::endl;
}
```