توثيق الـ Assembler البسيط (دعم ARM Assembly الموسع)

هذا المستند يوضح كيفية استخدام الـ assembler البسيط المكتوب بلغة C، والذي يدعم الآن مجموعة موسعة من تعليمات ARM Assembly، السجلات، والتسميات.

التعليمات المدعومة

بالإضافة إلى التعليمات السابقة، يدعم هذا الـ assembler الآن التعليمات التالية:

تعليمات معالجة البيانات (Data Processing Instructions)

- `MOV Rd, #imm` : نقل قيمة فورية إلى سجل. (مثال: `MOV R0, #10`)
- `MOV Rd, Rm` : نقل قيمة من سجل إلى سجل. (مثال: `MOV R9, R0`)
- `ADD Rd, Rn, #imm` : إضافة قيمة فورية إلى سجل وتخزين النتيجة في سجل آخر. (مثال: `ADD R2, R0, #3`)
- `ADD Rd, Rn, Rm` : إضافة سجل إلى سجل وتخزين النتيجة في سجل آخر. (مثال: `ADD R2, R0, R1`)
- `SUB Rd, Rn, #imm` : طرح قيمة فورية من سجل وتخزين النتيجة في سجل آخر. (مثال: `SUB R3, R2, #5`)
- `SUB Rd, Rn, Rm` : طرح سجل من سجل وتخزين النتيجة في سجل آخر. (مثال: `SUB R3, R2, R0`)
- `CMP Rn, #imm` : مقارنة سجل بقيمة فورية. (مثال: `CMP R3, #10`)
- `CMP Rn, Rm` : مقارنة سجل بسجل. (مثال: `CMP R3, R0`)
- `AND Rd, Rn, #imm` : عملية AND المنطقية بين سجل وقيمة فورية. (مثال: `AND R6, R0, #2`)
- `AND Rd, Rn, Rm` : عملية AND المنطقية بين سجلين. (مثال: `AND R6, R0, R1`)
- `ORR Rd, Rn, #imm` : عملية OR المنطقية بين سجل وقيمة فورية. (مثال: `ORR R7, R1, #1`)
- `ORR Rd, Rn, Rm` : عملية OR المنطقية بين سجلين. (مثال: `ORR R7, R1, R2`)

- `EOR Rd, Rn, #imm` : عملية XOR المنطقية بين سجل وقيمة فورية. (مثال: `EOR R8, R2, #4`)
- `EOR Rd, Rn, Rm` : عملية XOR المنطقية بين سجلين. (مثال: `EOR R8, R2, R3`)
- `LSL Rd, Rm, #imm` : إزاحة منطقية لليسار (Logical Shift Left). (مثال: `LSL R9, R0, #1`)
- `LSR Rd, Rm, #imm` : إزاحة منطقية لليمين (Logical Shift Right). (مثال: `LSR R10, R1, #1`)
- `MUL Rd, Rm, Rs` : ضرب سجلين وتخزين النتيجة في سجل. (مثال: `MUL R11, R0, R1`)

تعليمات القفز (Branch Instructions)

- `BGE label` : القفز إذا كان أكبر من أو يساوي. (مثال: `BGE label_high`)
- `BLT label` : القفز إذا كان أقل من. (مثال: `BLT loop`)
- `B label` : قفزة غير مشروطة. (مثال: `B loop_start`)

تعليمات استدعاء النظام (System Call Instruction)

- `SWI #imm` : استدعاء نظام. (مثال: `SWI #0`)

ملاحظات هامة:

- هذا الـ assembler هو تبسيط كبير لـ ARM Assembly. لا يدعم جميع أوضاع العنونة، ولا جميع التعليمات، ولا جميع السجلات (يدعم R0-R15 في هذا المثال، ولكن يجب أن تكون حذرًا مع استخدام السجلات العليا حيث أن الترميز مبسط جدًا).
- يتم التعامل مع المعاملات الفورية (immediate operands) كقيم عشرية بسيطة.
- يتم حساب عناوين القفز (branches) بشكل نسبي، ولكن بطريقة مبسطة جدًا.
- يتم دعم التسميات (Labels) التي تنتهي بـ `:` .

كيفية تجميع وتشغيل الـ Assembler

1. تجميع الكود المصدري

لتحويل الكود المصدري للـ assembler (ملف `assembler.c`) إلى ملف تنفيذي، استخدم مترجم GCC:

```bash
gcc assembler.c -o assembler
```

2. تشغيل الـ Assembler

بعد التجميع، يمكنك تشغيل الـ assembler باستخدام الصيغة التالية:

```bash
./assembler <input_file.text> <output_file.text>
```

- `<input_file.text>` : هو مسار ملف Assembly الذي يحتوي على التعليمات التي تريد تحويلها.
- `<output_file.text>` : هو مسار الملف الذي سيتم إنشاءه ويحتوي على الـ machine code المحول.

مثال على الاستخدام

لنفترض أن لديك ملف Assembly يسمى `arm_test_input_reg_ops.text` بالمحتوى التالي:

```plain text
_start:
    MOV     R0, #10
    MOV     R1, #3
    ADD     R2, R0, R1
    SUB     R3, R2, R0

    AND     R6, R0, R1
    ORR     R7, R1, R2
    EOR     R8, R2, R3

    MOV     R9, R0
    CMP     R3, R0

    LSL     R10, R0, #2
    LSR     R11, R1, #1
    MUL     R12, R0, R1

    B       end_program

end_program:
    SWI     #0
```

لتجميع هذا الملف، قم بتشغيل الأمر التالي:

```bash
./assembler arm_test_input_reg_ops.text arm_test_output_reg_ops.text
```

سيقوم هذا الأمر بإنشاء ملف `arm_test_output_reg_ops.text` بالمحتوى التالي (الـ machine code):

```text
E1A0000F
E1A0100F
E0802001
E0423000
E0006001
E1817002
E0228003
E1A09000
E1530000
E1A0A100
E1A0B0A1
E00C0190
EAFFFFFF
EF000000
```

هذه هي الأكواد السداسية عشرية المقابلة لتعليمات ARM Assembly المدخلة. كل سطر يمثل تعليمة واحدة.